ACL 2007

PRAGUE

IWPT 2007

Proceedings of the 10th International Conference on Parsing Technologies

June 23–34, 2007
Prague, Czech Republic

The Association for Computational Linguistics

Production and Manufacturing by
Omnipress
2600 Anderson Street
Madison, WI 53704
USA

©2007 Association for Computational Linguistics

Order copies of this and other ACL proceedings from:

Association for Computational Linguistics (ACL)
209 N. Eighth Street
Stroudsburg, PA 18360
USA
Tel: +1-570-476-8006
Fax: +1-570-476-0860
acl@aclweb.org

Preface

Welcome to the Tenth International Conference on Parsing Technologies, IWPT 2007, in the beautiful city of Prague.

IWPT'07 continues the tradition of biennial workshops on parsing technology organized by SIGPARSE, the Special Interest Group on Parsing of the Association for Computational Linguistics (ACL). The first workshop, in Pittsburgh and Hidden Valley, was followed by workshops in Cancun (Mexico) in 1991; Tilburg (Netherlands) and Durbuy (Belgium) in 1993; Prague and Karlovy Vary (Czech Republic) in 1995; Boston/Cambridge (Massachusetts) in 1997; Trento (Italy) in 2000; Beijing (China) in 2001; Nancy (France) in 2003; and Vancouver (Canada) in 2005.

Over the years the IWPT Workshops have become the major forum for researchers in natural language parsing. They have also given rise to four books on parsing technologies.

For the first time this year, IWPT is organised as a co-located event with the main ACL conference and with EMNLP and many other workshops. We would like to thank Alon Lavie, Priscilla Rasmussen and the ACL committee and local organisers for their help and support in organising this event.

Parsing technologies are relevant for almost all applications in Natural Language Processing. We are fortunate to have Stuart Shieber from Harvard University as our invited speaker to explore the links between sychronous grammars and issues related to machine translation and parsing.

This year's programme features for the first time invited presentations by organisers of co-located events who are also members of the IWPT Programme Committee. Joakim Nivre makes a connection with learning dependency grammars in the CONLL-07 shared task, and the organisers of the Deep Linguistic Processing workshop discuss the ways in which broad coverage parsing systems can be developed for linguistically expressive grammars.

I would to thank all the programme committee members for their careful and timely work, especially those that took up extra rewiewing obligations at very short notice. Special thanks go to Paola Merlo, the programme chair, for organising the reviewing, designing the workshop programme and producing the proceedings. The scientific programme includes 14 full papaer and 3 short papers out of 31 submissions (of which 6 short papers). They cover all topics in parsing, from efficiency issues and complextiy of algorithms to accurate supervised and unsupervised learning techniques for parsing.

Harry Bunt
IWPT 2007 General Chair

Organizers

General Chair:

Harry Bunt (Tilburg University, Netherlands)

Programme Chair:

Paola Merlo (University of Geneva, Switzerland)

Logistic Arrangements Chair:

Alon Lavie (Carnegie-Mellon University, Pittsburgh, USA)

Programme Committee:

Harry Bunt (Tilburg University, Netherlands)
David Chiang(USC/ISI,USA)
John Carroll (University of Sussex, Brighton, UK)
Stephen Clark (Oxford University, UK)
James Henderson (University of Edinburgh, UK)
Ulf Hermjakob (USC Information Sciences Institute, Marina del Rey, USA)
Julia Hockenmaier (University of Pennsylvania, USA)
Aravind Joshi (University of Pennsylvania, Philadelphia, USA)
Ronald Kaplan (Xerox Palo Alto Research Center, USA)
Martin Kay (Xerox Palo Alto Research Center, USA)
Sadao Kurohashi (University of Tokyo, Japan)
Alon Lavie (Carnegie-Mellon University, Pittsburgh, USA)
Rob Malouf (San Diego State University, USA)
Yuji Matsumoto (Nara Institute of Science and Technology, Japan)
Bob Moore (Microsoft, Redmond, USA)
Mark-Jan Nederhof (MPI, Groeningen, Netherlands)
Joakim Nivre (Vaxjo University, Sweden)
Gertjan van Noord (University of Groningen, Netherlands)
Stephan Oepen (University of Oslo, Norway)
Stefan Riezler (Xerox Palo Alto Research Center, USA)
Giorgio Satta (University of Padua, Italy)
Kenji Sagae (University of Tokyo, Japan)
Khalil Sima'an (University of Amsterdam, Netherlands)
Eric Villemonte de la Clergerie (INRIA, Rocquencourt, France)
K. Vijay-Shanker (University of Delaware, USA)
Dekai Wu (Hong Kong University of Science and Technology, China)

Invited Speaker:

Stuart Shieber, Harvard University

Co-located Event Spotlight Presenters:

Joakim Nivre (Vaxjo University, Sweden)
Organisers of the Deep Linguistic Processing Workshop

Table of Contents

Conference Program

Saturday, 23 June, 2007

9:00–9:35	Registration/Opening Remarks
9:35–10:10	*Using Self-Trained Bilexical Preferences to Improve Disambiguation Accuracy* Gertjan van Noord
10:10–10:45	*Evaluating Impact of Re-training a Lexical Disambiguation Model on Domain Adaptation of an HPSG Parser* Tadayoshi Hara, Yusuke Miyao and Jun'ichi Tsujii
Coffee break	
11:15–11:50	*Semi-supervised Training of a Statistical Parser from Unlabeled Partially-bracketed Data* Rebecca Watson, Ted Briscoe and John Carroll
11:50–12:05	*Adapting WSJ-Trained Parsers to the British National Corpus using In-Domain Self-Training* Jennifer Foster, Joachim Wagner, Djam Seddah and Josef van Genabith
	Co-located Event Spotlight Presentation
12:05–12:40	*The Impact of Deep Linguistic Processing on Parsing Technology* Timothy Baldwin, Mark Dras, Julia Hockenmaier, Tracy Holloway King and Gertjan van Noord
Lunch break	
14:00–14:35	*Improving the Efficiency of a Wide-Coverage CCG Parser* Bojan Djordjevic, James Curran and Stephen Clark
14:35–15:10	*Efficiency in Unification-Based N-Best Parsing* Yi Zhang, Stephan Oepen and John Carroll
15:10–15:45	*A log-linear model with an n-gram reference distribution for accurate HPSG parsing* Takashi Ninomiya, Takuya Matsuzaki, Yusuke Miyao and Jun'ichi Tsujii
Coffee break	

Saturday, 23 June, 2007 (continued)

16:15–16:50	*Ambiguity Resolution by Reordering Rules in Text Containing Errors* Sylvana Sofkova Hashemi
16:50–17:05	*Nbest Dependency Parsing with linguistically rich models* Xiaodong Shi
17:05–17:40	*Symbolic Preference Using Simple Scoring* Paula Newman

Sunday, 24 June, 2007

9:15–9:35	Registration
	Guest Speaker
9:30–10:45	*Synchronous Grammars and Transducers: Good News and Bad News* Stuart Shieber
Coffee break	
11:15–11:50	*Are Very Large Context-Free Grammars Tractable?* Pierre Boullier and Benot Sagot
11:50–12:05	*Pomset mcfgs* Michael Pan
12:05–12:40	*Modular and Efficient Top-Down Parsing for Ambiguous Left-Recursive Grammars* Richard Frost, Rahmatullah Hafiz and Paul Callaghan
Lunch break	
14:00–14:35	*On the Complexity of Non-Projective Data-Driven Dependency Parsing* Ryan McDonald and Giorgio Satta
14:35–15:10	*Dependency Parsing with Second-Order Feature Maps and Annotated Semantic Information* Massimiliano Ciaramita and Giuseppe Attardi

Using Self-Trained Bilexical Preferences to Improve Disambiguation Accuracy

Gertjan van Noord
University of Groningen
vannoord@let.rug.nl

Abstract

A method is described to incorporate bilexical preferences between phrase heads, such as selection restrictions, in a Maximum-Entropy parser for Dutch. The bilexical preferences are modelled as association rates which are determined on the basis of a very large parsed corpus (about 500M words). We show that the incorporation of such self-trained preferences improves parsing accuracy significantly.

1 Motivation

In parse selection, the task is to select the correct syntactic analysis of a given sentence from a set of parses generated by some other mechanism. On the basis of correctly labelled examples, supervised parse selection techniques can be employed to obtain reasonable accuracy. Although parsing has improved enormously over the last few years, even the most successful parsers make very silly, sometimes embarassing, mistakes. In our experiments with a large wide-coverage stochastic attribute-value grammar of Dutch, we noted that the system sometimes is insensitive to the naturalness of the various lexical combinations it has to consider. Although parsers often employ lexical features which are in principle able to represent preferences with respect to word combinations, the size of the training data will be too small to be able to learn the relevance of such features successfully.

In maximum-entropy parsing, the supervised parsing technique that we use in our experiments, arbitrary features can be defined which are employed to characterize different parses. So it is possible to construct features for any property that is thought to be important for disambiguation. However, such features can be useful for disambiguation only in case the training set contains a sufficient number of occurrences of these features. This is problematic, in practice, for features that encode bilexical preferences such as selection restrictions, because typical training sets are much too small to estimate the relevance of features representing cooccurrences of two words. As a simple example consider the ambiguous Dutch sentence

(1) Melk drinkt de baby niet
Milk drinks the baby not

The standard model of the parser we experimented with employs a wide variety of features including syntactic features and lexical features. In particular, the model also includes features which encode whether or not the subject or the object is fronted in a parse. Since subjects, in general, are fronted much more frequently than objects, the model has learnt to prefer readings in which the fronted constituent is analysed as the subject. Although the model also contains features to distinguish whether e.g. `milk` occurs as the subject or the object of `drink`, the model has not learnt a preference for either of these features, since there were no sentences in the training data that involved both these two words.

To make this point more explicit, we found that in about 200 sentences of our parsed corpus of 27 million sentences `milk` is the head of the direct object of the verb `drink`. Suppose that we would need at least perhaps 5 to 10 sentences in our training corpus in order to be able to learn the specific preference between `milk` and `drink`. The implication is that we would need a (manually labeled!) training corpus of approximately 1 million sentences (20 million words). In contrast, the disambiguation model of the Dutch parser we are reporting on in this paper is trained on a manually labeled corpus of slightly over 7,000 sentences (145,000 words). It appears that semi-supervised or un-supervised methods are required here.

Note that the problem not only occurs for artificial examples such as (1); here are a few mis-parsed examples actually encountered in a large parsed corpus:

(2) a. Campari moet **u** gedronken hebben
Campari must have drunk you
You must have drunk Campari

b. De wijn die **Elvis** zou hebben gedronken als hij wijn zou hebben gedronken
The wine Elvis would have drunk if he had drunk wine
The wine that would have drunk Elvis if he had drunk wine

c. De paus heeft **tweehonderd daklozen** te eten gehad
The pope had twohunderd homeless people for dinner

In this paper, we describe an alternative approach in which we employ pointwise mutual information association score in the maximum entropy disambiguation model. Pointwise mutual information (Fano, 1961) was used to measure strength of selection restrictions for instance by Church and Hanks (1990). The association scores used here are estimated using a very large parsed corpus of 500 million words (27 million sentences). We show that the incorporation of this additional knowledge source improves parsing accuracy. Because the association scores are estimated on the basis of a large corpus that is parsed by the parser that we aim to improve upon, this technique can be described as a somewhat particular instance of self-training. Self-training has been investigated for statistical parsing before. Although naively adding self-labeled material to extend training data is normally not succesfull, there have been successful variants of self-learning for parsing as well. For instance, in McClosky et al. (2006) self-learning is used to improve a two-phase parser reranker, with very good results for the classical Wall Street Journal parsing task.

Clearly, the idea that selection restrictions ought to be useful for parsing accuracy is not new. However, as far as we know this is the first time that automatically acquired selection restrictions have been shown to improve parsing accuracy results. Related research includes Abekawa and Okumura (2006) and Kawahara and Kurohashi (2006) where statistical information between verbs and case elements is collected on the basis of large automatically analysed corpora.

2 Background: Alpino parser

The experiments are performed using the Alpino parser for Dutch. In this section we briefly describe the parser, as well as the corpora that we have used in the experiments described later.

2.1 Grammar and Lexicon

The Alpino system is a linguistically motivated, wide-coverage grammar and parser for Dutch in the tradition of HPSG. It consists of over 600 grammar rules and a large lexicon of over 100,000 lexemes and various rules to recognize special constructs such as named entities, temporal expressions, etc. The grammar takes a ‘constructional’ approach, with rich lexical representations and a large number of detailed, construction specific rules. Both the lexicon and the rule component are organized in a multiple inheritance hierarchy. Heuristics have been implemented to deal with unknown words and word sequences, and ungrammatical or out-of-coverage sentences (which may nevertheless contain fragments that are analysable). The Alpino system includes a POS-tagger which greatly reduces lexical ambiguity, without an observable decrease in parsing accuracy (Prins, 2005).

2.2 Parser

Based on the categories assigned to words, and the set of grammar rules compiled from the HPSG grammar, a left-corner parser finds the set of all parses, and stores this set compactly in a packed parse forest. All parses are rooted by an instance of the top category, which is a category that generalizes over all maximal projections (S, NP, VP, ADVP, AP, PP and some others). If there is no parse covering the complete input, the parser finds all parses for each substring. In such cases, the robustness component will then select the best sequence of non-overlapping parses (i.e., maximal projections) from this set.

In order to select the best parse from the compact parse forest, a best-first search algorithm is applied. The algorithm consults a Maximum Entropy disambiguation model to judge the quality of (partial) parses. Since the disambiguation model includes inherently non-local features, efficient dynamic programming solutions are not directly applicable. Instead, a best-first beam-search algorithm is employed (van Noord and Malouf, 2005; van Noord, 2006).

2.3 Maximum Entropy disambiguation model

The maximum entropy model is a conditional model which assigns a probability to a parse t for a given sentence s. Furthermore, $f_i(t)$ are the feature functions which count the occurrence of each feature i in a parse t. Each feature i has an associated weight λ_i. The score ϕ of a parse t is defined as the sum of the weighted feature counts:

$$\phi(t) = \sum_i \lambda_i f_i(t)$$

If t is a parse of s, the actual conditional probability is given by the following, where $T(s)$ are all parses of s:

$$P(t|s) = \frac{\exp(\phi(t))}{\sum_{u \in T(s)} \exp(\phi(u))}$$

However, note that if we only want to select the best parse we can ignore the actual probability, and it suffices to use the score ϕ to rank competing parses.

The Maximum Entropy model employs a large set of features. The standard model uses about 42,000 different features. Features describe various properties of parses. For instance, the model includes features which signal the application of particular grammar rules, as well as local configurations of grammar rules. There are features signalling specific POS-tags and subcategorization frames. Other features signal local or non-local occurrences of extraction (WH-movement, relative clauses etc.), the grammatical role of the extracted element (subject vs. non-subject etc.), features to represent the distance of a relative clause and the noun it modifies, features describing the amount of parallelism between conjuncts in a coordination, etc. In addition, there are lexical features which represent the co-occurrence of two specific words in a specific dependency, and the occurrence of a specific word as a specific dependent for a given POS-tag. Each parse is characterized by its feature vector (the counts for each of the 42,000 features). Once the model is trained, each feature is associated with its weight λ (a positive or negative number, typically close to 0). To find out which parse is the best parse according to the model, it suffices to multiply the frequency of each feature with its corresponding weight, and sum these weighted frequencies. The parse with the highest sum is the best parse. Formal details of the disambiguation model are presented in van Noord and Malouf (2005).

2.4 Dependency structures

Although Alpino is not a dependency grammar in the traditional sense, dependency structures are generated by the lexicon and grammar rules as the value of a dedicated feature `dt`. The dependency structures are based on CGN (Corpus Gesproken Nederlands, Corpus of Spoken Dutch) (Hoekstra et al., 2003), D-Coi and LASSY (van Noord et al., 2006). Such dependency structures are somewhat idiosyncratic, as can be observed in the example in figure 1 for the sentence:

(3) waar en wanneer dronk Elvis wijn?
where and when did Elvis drink wine?

2.5 Evaluation

The output of the parser is evaluated by comparing the generated dependency structure for a corpus sentence to the gold standard dependency structure in a treebank. For this comparison, we represent the dependency structure (a directed acyclic graph) as a set of named dependency relations. The dependency graph in figure 1 is represented with the following set of dependencies:

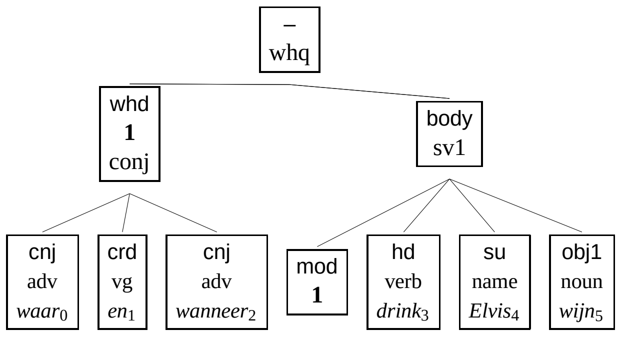


Figure 1: Dependency graph example. Reentrant nodes are visualized using a bold-face index. Root forms of head words are explicitly included in separate nodes, and different types of head receive a different relation label such as `hd`, `crd` (for coordination), `whd` (for WH-phrases) etc. In this case, the WH-phrase is both the *whd* element of the top-node, as well as a *mod* dependent of *drink*.

crd/cnj(`en`, `waar`)	crd/cnj(`en`, `wanneer`)
whd/body(`en`, `drink`)	hd/mod(`drink`, `en`)
hd/obj1(`drink`, `wijn`)	hd/su(`drink`, `Elvis`)

Comparing these sets, we count the number of dependencies that are identical in the generated parse and the stored structure, which is expressed traditionally using f-score (Briscoe et al., 2002). We prefer to express similarity between dependency structures by *concept accuracy*:

$$\text{CA} = 1 - \frac{\sum_i D_f^i}{\max(\sum_i D_g^i, \sum_i D_p^i)}$$

where D_p^i is the number of dependencies produced by the parser for sentence i, D_g is the number of dependencies in the treebank parse, and D_f is the number of incorrect and missing dependencies produced by the parser.

The standard version of Alpino that we use here as baseline system is trained on the 145,000 word Alpino treebank, which contains dependency structures for the `cdbl` (newspaper) part of the Eindhoven corpus. The parameters for training the model are the same for the baseline model, as well as the model that includes the self-trained bilexical preferences (introduced below). These parameters include the Gaussian penalty, thresholds for feature selection, etc. Details of the training procedure are described in van Noord and Malouf (2005).

#sentences	100%	30,000,000
#words		500,000,000
#sentences without parse	0.2%	100,000
#sentences with fragments	8%	2,500,000
#single full parse	92%	27,500,000

Table 1: Approximate counts of the number of sentences and words in the parsed corpus. About 0,2% of the sentences did not get a parse, for computational reasons (out of memory, or maximum parse time exceeded).

2.6 Parsed Corpora

Over the course of about a year, Alpino has been used to parse most of the TwNC-02 (Twente Newspaper Corpus), Dutch Wikipedia, and the Duch part of Europarl. TwNC consists of Dutch newspaper texts from 1994 - 2004. We did not use the material from Trouw 2001, since part of that material is used in the test set used below. We used the 200 node Beowulf Linux cluster of the High-Performance Computing center of the University of Groningen. The dependency structures are stored in XML. The XML files can be processed and searched in various ways, for instance, using XPATH, XSLT and Xquery (Bouma and Kloosterman, 2002). Some quantitative information of this parsed corpus is listed in table 1. In the experiments described below, we do not distinguish between full and fragment parses; sentences without a parse are obviously ignored.

3 Bilexical preferences

3.1 Association Score

The parsed corpora described in the previous section have been used in order to compute association scores between lexical dependencies. The parses constructed by Alpino are dependency structures. In such dependency structures, the basic dependencies are of the form $r(w_1, w_2)$ where r is a relation such as *subject, object, modifier, prepositional complement, . . .*, and w_i are root forms of words.

Bilexical preference between two root forms w_1

tokens	480,000,000
types	100,000,000
types with frequency ≥ 20	2,000,000

Table 2: Number of lexical dependencies in parsed corpora (approximate counts)

bijltje	gooi_neer	13
duimschroef	draai_aan	13
peentje	zweet	13
traantje	pink_weg	13
boontje	dop	12
centje	verdien_bij	12
champagne_fles	ontkurk	12
dorst	les	12

Table 3: Pairs involving a direct object relationship with the highest pointwise mutual information score.

and w_2 is computed using an association score based on *pointwise mutual information*, as defined by Fano (1961) and used for a similar purpose in Church and Hanks (1990), as well as in many other studies in corpus linguistics. The association score is defined here as follows:

$$I(r(w_1, w_2) = \log \frac{f(r(w_1, w_2))}{f(r(w_1, \_))f(\_(\_, w_2))}$$

where $f(X)$ is the relative frequency of X. In the above formula, the underscore is a place holder for an arbitrary relation or an arbitrary word. The association score I compares the actual relative frequency of w_1 and w_2 with dependency r, with the relative frequency we would expect if the words were independent. For instance, to compute I(hd/obj1(`drink`,`melk`)) we lookup the number of times `drink` occurs with a direct object out of all 462,250,644 dependencies (15,713) and the number of times `melk` occurs as a dependent (10,172). If we multiply the two corresponding relative frequencies, we get the expected relative frequency (0.35) for hd/obj1(`drink`,`melk`), which is about 560 times as big as the actual frequence, 195. Taking the log of this gives us the association score (6.33) for this bi-lexical dependency. Note that pairs that we have seen fewer than 20 times are ignored. Mutual information scores are unreliable for low frequencies. An additional benefit of a frequency threshold is a manageable size of the resulting data-structures.

The pairs involving a direct object relationship with the highest scores are listed in table 3. The highest scoring nouns that occur as the direct object of `drink` are listed in table 4.

biertje	small glass of beer	8
borreltje	strong alcoholic drink	8
glaasje	small glass	8
pilsje	small glass of beer	8
pintje	small glass of beer	8
pint	glass of beer	8
wijntje	small glass of wine	8
alcohol	alcohol	7
bier	beer	7

Table 4: Pairs involving a direct object relationship with the highest pointwise mutual information score for the verb `drink`.

overlangs	snijd_door	12
welig	tier	12
dunnetjes	doe_over	11
stief_moederlijk	bedeel	11
on_zedelijk	betast	11
stierlijk	verveel	11
cum laude	studeer_af	10
hermetisch	grendel_af	10
ingespannen	tuur	10
instemmend	knik	10
kostelijk	amuseer	10

Table 5: Pairs involving a modifier relationship between a verb and an adverbial with the highest association score.

Selection restrictions are often associated only with direct objects. We include bilexical association scores for all types of dependencies. We found that association scores for other types of dependencies also captures both collocational preferences as well as weaker cooccurrence preferences. Some examples including modifiers are listed in table 5. Such preferences are useful for disambiguation as well. Consider the ambiguous Dutch sentence

(4) omdat we lauw bier dronken
because we drank warm beer
because we drank beer warmly

The adjective `lauw` (cold, lukewarm, warm) can be used to modify both nouns and verbs; this latter possibility is exemplified in:

(5) We hebben lauw gereageerd
We reacted indifferently

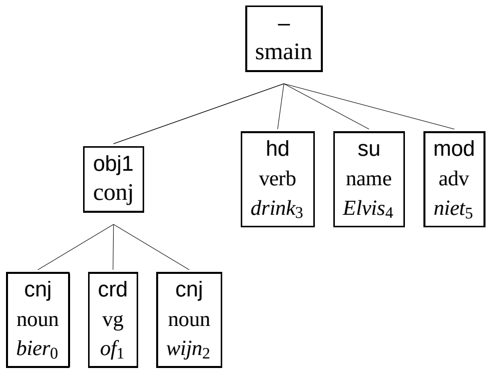


Figure 2: Dependency structure produced for coordination

3.2 Extending pairs

The CGN dependencies that we work with fail to relate pairs of words in certain syntactic constructions for which it can be reasonably assumed that bilexical preferences should be useful. We have identified two such constructions, namely relative clauses and coordination, and for these constructions we generalize our method, to take such dependencies into account too.

Consider coordinations such as:

(6) Bier of wijn drinkt Elvis niet
Beer or wine, Elvis does not drink

The dependency structure of the intended analysis is given in figure 2. The resulting set of dependencies for this example treats the coordinator as the head of the conjunction:

hd/obj1(`drink`, `of`)	crd/cnj(`of`, `bier`)
crd/cnj(`of`, `wijn`)	hd/su(`drink`, `elvis`)
hd/mod(`drink`, `niet`)	

So there are no direct dependencies between the verb and the individual conjuncts. For this reason, we add additional dependencies $r(A, C)$ for every pair of dependency $r(A, B)$, crd/cnj(B, C).

Relative clauses are another syntactic phenomenon where we extend the set of dependencies. For a noun phrase such as:

(7) Wijn die Elvis niet dronk
Wine which Elvis did not drink

there is no direct dependency between `wijn` and `drink`, as can be seen in the dependency structure

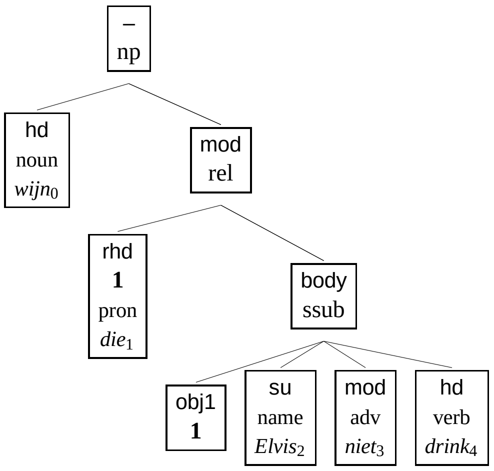


Figure 3: Dependency structure produced for relative clause

given in figure 3. Sets of dependencies are extended in such cases, to make the relation between the noun and the role it plays in the relative clause explicit.

3.3 Using association scores as features

The association scores for all dependencies are used in our maximum entropy disambiguation model as follows. The technique is reminiscent of the inclusion of auxiliary distributions in stochastic attribute-value grammar (Johnson and Riezler, 2000).

Recall that a maximum entropy disambiguation model exploits features. Features are properties of parses, and we can use such features to describe any property of parses that we believe is of importance for disambiguation. For the disambiguation model, a parse is fully characterized by a vector of feature counts.

We introduce features $z(t, r)$ for each of the major POS labels t (verb, noun, adjective, adverb, ...) and each of the dependency relations r. The 'count' of such a feature is determined by the association scores for actually occuring dependency pairs. For example, if in a given parse a given verb v has a direct object dependent n, then we compute the association of this particular pair, and use the resulting number as the count of that feature. Of course, if there are multiple dependencies of this type in a single parse, the corresponding association scores are all summed.

To illustrate this technique, consider the dependency structure given earlier in figure 2. For this example, there are four of these new features with a non-zero count. The counts are given by the corresponding association scores as follows:

$$\begin{array}{lcl} z(\text{verb}, \text{hd/su}) & = & I(\text{hd/su}(\texttt{drink}, \texttt{elvis})) \\ z(\text{verb}, \text{hd/mod}) & = & I(\text{hd/mod}(\texttt{drink}, \texttt{niet})) \\ z(\text{verb}, \text{hd/obj1}) & = & I(\text{hd/obj1}(\texttt{drink}, \texttt{of})) \\ & + & I(\text{hd/obj1}(\texttt{drink}, \texttt{bier})) \\ & + & I(\text{hd/obj1}(\texttt{drink}, \texttt{wijn})) \\ z(\text{conj}, \text{crd/cnj}) & = & I(\text{crd/cnj}(\texttt{of}, \texttt{bier})) \\ & + & I(\text{crd/cnj}(\texttt{of}, \texttt{wijn})) \end{array}$$

It is crucial to observe that the new features do not include any direct reference to actual words. This means that there will be only a fairly limited number of new features (depending on the number of tags t and relations r), and we can expect that these features are frequent enough to be able to estimate their weights in training material of limited size.

Association scores can be negative if two words in a lexical dependency occur less frequently than one would expect if the words were independent. However, since association scores are unreliable for low frequencies (including, often, frequencies of zero), and since such negative associations involve low frequencies by their nature, we only take into account positive association scores.

4 Experiments

We report on two experiments. In the first experiment, we report on the results of tenfold cross-validation on the Alpino treebank. This is the material that is standardly used for training and testing. For each of the sentences of this corpus, the system produces atmost the first 1000 parses. For every parse we compute the quality by comparing its dependency structure with the gold standard dependency structure in the treebank. For training, atmost 100 parses are selected randomly for each sentence. For (tenfold cross-validated) testing, we use all available parses for a given sentence. In order to test the quality of the model, we check for each given sentence which of its atmost 1000 parses is selected by the disambiguation model. The quality of that parse is used in the computation of the accuracy, as listed in table 6. The column labeled *exact* measures the proportion of sentences for which the model selected the best possible parse (there can be multiple best possible parses). The *baseline* row reports on the quality of a disambiguation model which simply selects the first parse for each sentence. The *oracle* row reports on the quality of the best-possible disambiguation model, which would (by magic) always select the best possible parse (some parses are outside the coverage of the system, and some parses are generated only after more than 1000 inferior parses). The *error reduction* column measures which part of the disambiguation problem (difference between the baseline and oracle scores) is solved by the model.[1]

	fscore	err.red.	exact	CA
	%	%	%	%
baseline	74.02	0.00	16.0	73.48
oracle	91.97	100.00	100.0	91.67
standard	87.41	74.60	52.0	87.02
+self-training	87.91	77.38	54.8	87.51

Table 6: Results with ten-fold cross-validation on the Eindhoven-cdbl part of the Alpino treebank. In these experiments, the models are used to select a parse from a given set of atmost 1000 parses per sentence.

The results show a small but clear increase in error reduction, if the standard model (without the association score features) is compared with a (retrained) model that includes the association score features. The relatively large improvement of the *exact* score suggests that the bilexical preference features are particularly good at choosing between very good parses.

For the second experiment, we evaluate how well the resulting model performs in the full system. First of all, this is the only really convincing evaluation which measures progress for the system as a whole by virtue of including bilexical preferences. The second motivation for this experiment is for methodological reasons: we now test on a truly unseen test-set. The first experiment can be criti-

[1]Note that the error reduction numbers presented in the table are lower than those presented in van Noord and Malouf (2005). The reason is, that we report here on experiments in which parses are generated with a version of Alpino with the POS-tagger switched on. The POS-tagger already reduces the number of ambiguities, and in particular solves many of the 'easy' cases. The resulting models, however, are more effective in practice (where the model also is applied after the POS-tagger).

	prec %	rec %	fscore %	CA %
standard	90.77	90.49	90.63	90.32
+self-training	91.19	90.89	91.01	90.73

Table 7: Results on the WR-P-P-H part of the D-Coi corpus (2267 sentences from the newspaper Trouw, from 2001). In these experiments, we report on the full system. In the full system, the disambiguation model is used to guide a best-first beam-search procedure which extracts a parse from the parse forest. Difference in CA was found to be significant (using paired T-test on the per sentence CA scores).

cized on methodological grounds as follows. The Alpino Treebank was used to train the disambiguation model which was used to construct the large parsed treebank from which we extracted the counts for the association scores. Those scores might somehow therefore indirectly reflect certain aspects of the Alpino Treebank training data. Testing on that data later (with the inclusion of the association scores) is therefore not sound.

For this second experiment we used the WR-P-P-H (newspaper) part of the D-Coi corpus. This part contains 2256 sentences from the newspaper Trouw (2001). In table 7 we show the resulting f-score and CA for a system with and without the inclusion of the $z(t, r)$ features. The improvement found in the previous experiment is confirmed.

5 Conclusion and Outlook

One might wonder why self-training works in the case of selection restrictions, at least in the set-up described above. One may argue that, in order to learn that *milk* is a good object for *drink*, the parser has to analyse examples of *drink milk* in the raw data correctly. But if the parser is capable of analysing these examples, why does it need selection restrictions? The answer appears to be that the parser (without selection restrictions) is able to analyse the large majority of cases correctly. These cases include the many easy occurrences where no (difficult) ambiguities arise (case marking, number agreement and other syntactic characteristics often force a single reading). The easy cases outnumber the misparsed difficult cases, and therefore the selection restrictions can be learned. Using these selection restrictions as additional features, the parser is then able to also get the difficult, ambiguous, cases right.

There are various aspects of our method that need further investigation. First of all, existing techniques that involve selection restrictions (e.g., Resnik (1993)) typically assume classes of nouns, rather than individual nouns. In future work we hope to generalize our method to take classes into account, where the aim is to learn class membership also on the basis of large parsed corpora.

Another aspect of the technique that needs further research involves the use of a threshold in establishing the association score, and perhaps related to this issue, the incorporation of negative association scores (for instance for cases where a large number of cooccurrences of a pair would be expected but where in fact none or very few were found).

There are also some more practical issues that perhaps had a negative impact on our results. First, the large parsed corpus was collected over a period of about a year, but during that period, the actual system was not stable. In particular, due to various improvements of the dictionary, the root form of words that was used by the system changed over time. Since we used root forms in the computation of the association scores, this could be harmful in some specific cases. A further practical issue concerns repeated sentences or even full paragraphs. This happens in typical newspaper material for instance in the case of short descriptions of movies that may be repeated weekly for as long as that movie is playing. Pairs of words that occur in such repeated sentences receive association scores that are much too high. The method should be adapted to take this into account, perhaps simply by removing duplicated sentences.

Clearly, the idea that selection restrictions ought to be useful for parsing is not new. However, as far as we know this is the first time that automatically acquired selection restrictions have been shown to improve parsing accuracy results.

Acknowledgements

This research was carried out in part in the context of the D-Coi and Lassy projects. The D-Coi and Lassy projects are carried out within the STEVIN programme which is funded by the Dutch and Flemish governments (http://taalunieversum.org/taal/technologie/stevin/).

References

Takeshi Abekawa and Manabu Okumura. 2006. Japanese dependency parsing using co-occurrence information and a combination of case elements. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 833–840, Sydney, Australia, July. Association for Computational Linguistics.

Gosse Bouma and Geert Kloosterman. 2002. Querying dependency treebanks in XML. In *Proceedings of the Third international conference on Language Resources and Evaluation (LREC)*, pages 1686–1691, Gran Canaria, Spain.

Ted Briscoe, John Carroll, Jonathan Graham, and Ann Copestake. 2002. Relational evaluation schemes. In *Proceedings of the Beyond PARSEVAL Workshop at the 3rd International Conference on Language Resources and Evaluation*, pages 4–8, Las Palmas, Gran Canaria.

Kenneth Ward Church and Patrick Hanks. 1990. Word association norms, mutual information and lexicography. *Computational Linguistics*, 16(1):22–29.

Robert Mario Fano. 1961. *Transmission of Information: A Statistical Theory of Communications*. MIT Press, Cambridge, MA.

Heleen Hoekstra, Michael Moortgat, Bram Renmans, Machteld Schouppe, Ineke Schuurman, and Ton van der Wouden, 2003. *CGN Syntactische Annotatie*, December.

Mark Johnson and Stefan Riezler. 2000. Exploiting auxiliary distributions in stochastic unification-based grammars. In *Proceedings of the first conference on North American chapter of the Association for Computational Linguistics*, pages 154–161, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Daisuke Kawahara and Sadao Kurohashi. 2006. A fully-lexicalized probabilistic model for japanese syntactic and case structure analysis. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 176–183, Morristown, NJ, USA. Association for Computational Linguistics.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 152–159, New York City, USA, June. Association for Computational Linguistics.

Robbert Prins. 2005. *Finite-State Pre-Processing for Natural Language Analysis*. Ph.D. thesis, University of Groningen.

Philip Stuart Resnik. 1993. *Selection and information: a class-based approach to lexical relationships*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, USA.

Gertjan van Noord and Robert Malouf. 2005. Wide coverage parsing with stochastic attribute value grammars. Draft available from http://www.let.rug.nl/˜vannoord. A preliminary version of this paper was published in the Proceedings of the IJCNLP workshop Beyond Shallow Analyses, Hainan China, 2004.

Gertjan van Noord, Ineke Schuurman, and Vincent Vandeghinste. 2006. Syntactic annotation of large corpora in STEVIN. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, Genoa, Italy.

Gertjan van Noord. 2006. **A**t **L**ast **P**arsing **I**s **N**ow **O**perational. In *TALN 2006 Verbum Ex Machina, Actes De La 13e Conference sur Le Traitement Automatique des Langues naturelles*, pages 20–42, Leuven.

Examples

Here we list a number of examples, which suggest that selection restrictions can also be important for dependencies, other than direct objects.

High scoring pairs involving a subject relationship with a verb:

alarmbel	rinkel
champagnekurk	knal
gij	echtbreek
haan	kraai
kikker	kwaak
rups	verpop
vonk	overspring
zweet	parel
belletje	rinkel
brievenbus	klepper

High scoring pairs involving a modifier relationship with a noun:

in vitro	fertilisatie
Hubble	ruimtetelescoop
zelfrijzend	bakmeel
bezittelijk	voornaamwoord
ingegroeid	teennagel
knapperend	haardvuur
levendbarend	hagedis
onbevlekt	ontvangenis
ongeblust	kalk

High scoring pairs involving a predicative complement relationship with a verb:

beetgaar	kook
beuk	murw
schuimig	klop
suf	peins
suf	pieker
doormidden	scheur
ragfijn	hak
stuk	bijt
au serieux	neem
in duigen	val
lam	leg

High scoring pairs involving an apposition relationship with a noun:

jongensgroep	Boyzone
communicatiesysteem	C2000
blindeninstituut	De Steffenberg
haptonoom	Ted Troost
gebedsgenezeres	Greet Hofmans
rally	Parijs-Dakar
tovenaar	Gandalf
aartsengel	Gabriel
keeperstrainer	Joep Hiele
basketbalcoach	Ton Boot
partizaan	Tito

High scoring pairs involving a measure phrase relationship with an adjective:

graadje	erger
lichtjaar	verwijderd
mijlenver	verwijderd
niets	liever
eindje	verderop
graad	warmer
illusie	armer
kilogram	wegend
onsje	minder
maatje	te groot
knip	waard

Evaluating Impact of Re-training a Lexical Disambiguation Model on Domain Adaptation of an HPSG Parser

Tadayoshi Hara[1] **Yusuke Miyao**[1] **Jun'ichi Tsujii**[1,2,3]
[1]Department of Computer Science, University of Tokyo
Hongo 7-3-1, Bunkyo-ku, Tokyo 113-0033 Japan
[2]School of Computer Science, University of Manchester
POBox 88, Sackville St, MANCHESTER M60 1QD, UK
[3]NaCTeM(National Center for Text Mining)
Manchester Interdisciplinary Biocentre, University of Manchester
131 Princess St, MANCHESTER M1 7DN, UK
E-mail: {harasan, yusuke, tsujii}@is.s.u-tokyo.ac.jp

Abstract

This paper describes an effective approach to adapting an HPSG parser trained on the Penn Treebank to a biomedical domain. In this approach, we train probabilities of lexical entry assignments to words in a target domain and then incorporate them into the original parser. Experimental results show that this method can obtain higher parsing accuracy than previous work on domain adaptation for parsing the same data. Moreover, the results show that the combination of the proposed method and the existing method achieves parsing accuracy that is as high as that of an HPSG parser retrained from scratch, but with much lower training cost. We also evaluated our method in the Brown corpus to show the portability of our approach in another domain.

1 Introduction

Domain portability is an important aspect of the applicability of NLP tools to practical tasks. Therefore, domain adaptation methods have recently been proposed in several NLP areas, e.g., word sense disambiguation (Chan and Ng, 2006), statistical parsing (Lease and Charniak, 2005; McClosky et al., 2006), and lexicalized-grammar parsing (Johnson and Riezler, 2000; Hara et al., 2005). Their aim was to re-train a probabilistic model for a new domain at low cost, and more or less successfully improved the accuracy for the domain.

In this paper, we propose a method for adapting an HPSG parser (Miyao and Tsujii, 2002; Ninomiya et al., 2006) trained on the WSJ section of the Penn Treebank (Marcus et al., 1994) to a biomedical domain. Our method re-trains a probabilistic model of lexical entry assignments to words in a target domain, and incorporates it into the original parser. The model of lexical entry assignments is a log-linear model re-trained with machine learning features only of word n-grams. Hence, the cost for the re-training is much lower than the cost of training the entire disambiguation model from scratch.

In the experiments, we used an HPSG parser originally trained with the Penn Treebank, and evaluated a disambiguation model re-trained with the GENIA treebank (Kim et al., 2003), which consists of abstracts of biomedical papers. We varied the size of a training corpus, and measured the transition of the parsing accuracy and the cost required for parameter estimation. For comparison, we also examined other possible approaches to adapting the same parser. In addition, we applied our approach to the Brown corpus (Kucera and Francis, 1967) in order to examine portability of our approach.

The experimental results revealed that by simply re-training the probabilistic model of lexical entry assignments we achieve higher parsing accuracy than with a previously proposed adaptation method. In addition, combined with the existing adaptation method, our approach achieves accuracy as high as that obtained by re-training the original parser from scratch, but with much lower training cost. In this paper, we report these experimental results in detail, and discuss how disambiguation models of lexical entry assignments contribute to domain adaptation.

In recent years, it has been shown that lexical information plays a very important role for high accuracy of lexicalized grammar parsing. Bangalore and Joshi (1999) indicated that, correct disambiguation with supertagging, i.e., assignment of lexical entries before parsing, enabled effective LTAG (Lexicalized Tree-Adjoining Grammar) parsing. Clark and Curran (2004a) showed that supertagging reduced cost for training and execution of a CCG (Combinatory Categorial Grammar) parser while keeping accuracy. Clark and Curran (2006) showed that a CCG parser trained on data derived from lexical category sequences alone was only slightly less accurate than one trained on complete dependency structures. Ninomiya et al. (2006) also succeeded in significantly improving speed and accuracy of HPSG parsing by using supertagging probabilities. These results indicate that the probability of lexical entry assignments is essential for parse disambiguation.

Such usefulness of lexical information has also been shown for domain adaptation methods. Lease and Charniak (2005) showed how existing domain-specific lexical resources on a target domain may be leveraged to augment PTB-training: part-of-speech tags, dictionary collocations, and named-entities. Our findings basically follow the above results. The contribution of this paper is to provide empirical results of the relationships among domain variation, probability of lexical entry assignment, training data size, and training cost. In particular, this paper empirically shows how much in-domain corpus is required for satisfiable performance.

In Section 2, we introduce an HPSG parser and describe an existing method for domain adaptation. In Section 3, we show our methods of re-training a lexical disambiguation model and incorporating it into the original model. In Section 4, we examine our method through experiments on the GENIA treebank. In Section 5, we examine the portability of our method through experiments on the Brown corpus. In Section 6, we showed several recent researches related to domain adaptation.

2 An HPSG Parser

HPSG (Pollard and Sag, 1994) is a syntactic theory based on lexicalized grammar formalism. In HPSG, a small number of grammar rules describe general construction rules, and a large number of

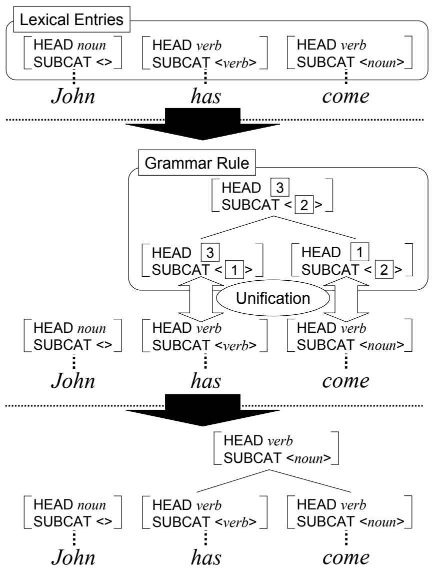


Figure 1: Parsing a sentence “*John has come.*”

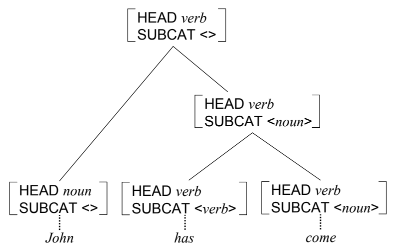


Figure 2: An HPSG parse tree for a sentence “*John has come.*”

lexical entries express word-specific characteristics. The structures of sentences are explained using combinations of grammar rules and lexical entries.

Figure 1 shows an example of HPSG parsing of the sentence “*John has come.*” First, as shown at the top of the figure, an HPSG parser assigns a lexical entry to each word in this sentence. Next, a grammar rule is assigned and applied to lexical entries. At the middle of this figure, the grammar rule is applied to the lexical entries for “*has*” and “*come.*” We then obtain the structure represented at the bottom of the figure. After that, the application of grammar rules is done iteratively, and then we can finally obtain the parse tree as is shown in Figure 2. In practice, since two or more parse candidates can be given for one sentence, a disambiguation model gives probabilities to these candidates, and a candidate given the highest probability is then chosen as a correct parse.

The HPSG parser used in this study is Ninomiya et al. (2006), which is based on *Enju* (Miyao and Tsujii, 2005). Lexical entries of Enju were extracted from the Penn Treebank (Marcus et al., 1994), which consists of sentences collected from The Wall Street Journal (Miyao et al., 2004). The disambiguation model of Enju was trained on the same treebank.

The disambiguation model of Enju is based on a feature forest model (Miyao and Tsujii, 2002), which is a log-linear model (Berger et al., 1996) on packed forest structure. The probability, $p_E(t|\mathbf{w})$, of producing the parse result t for a given sentence $\mathbf{w} = \langle w_1, ..., w_u \rangle$ is defined as

$$p_E(t|\mathbf{w}) = \frac{1}{Z_s} \prod_i p_{lex}(l_i|\mathbf{w}, i) \cdot q_{syn}(t|\mathbf{l}),$$

$$Z_s = \sum_{t \in T(\mathbf{w})} \prod_i p_{lex}(l_i|\mathbf{w}, i) \cdot q_{syn}(t|\mathbf{l})$$

where $\mathbf{l} = \langle l_1, ..., l_u \rangle$ is a list of lexical entries assigned to $\mathbf{w}$, $p_{lex}(l_i|\mathbf{w}, i)$ is a probabilistic model giving the probability that lexical entry l_i is assigned to word w_i, $q_{syn}(t|\mathbf{l})$ is an unnormalized log-linear model of tree construction and gives the possibility that parse candidate t is produced from lexical entries $\mathbf{l}$, and $T(\mathbf{w})$ is a set of parse candidates assigned to $\mathbf{w}$. With a treebank of a target domain as training data, model parameters of p_{lex} and q_{syn} are estimated so as to maximize the log-likelihood of the training data.

Probabilistic model p_{lex} is defined as a log-linear model as follows.

$$p_{lex}(l_i|\mathbf{w}, i) = \frac{1}{Z_{w_i}} \exp\left(\sum_j \lambda_j f_j(l_i, \mathbf{w}, i)\right),$$

$$Z_{w_i} = \sum_{l_i \in L(w_i)} \exp\left(\sum_j \lambda_j f_j(l_i, \mathbf{w}, i)\right),$$

where $L(w_i)$ is a set of lexical entries which can be assigned to word w_i. Before training this model, $L(w_i)$ for all w_i are extracted from the training treebank. The feature function $f_j(l_i, \mathbf{w}, i)$ represents the characteristics of l_i, $\mathbf{w}$ and w_i, while corresponding λ_j is its weight. For the feature functions, instead of using unigram features adopted in Miyao and Tsujii (2005), Ninomiya et al. (2006) used "word trigram" and "POS 5-gram" features which are listed in Table 1. With the revised Enju model, they achieved parsing accuracy as high as Miyao and Tsujii (2005), with around four times faster parsing speed.

Table 1: Features for the probabilities of lexical entry selection

surrounding words	$w_{-1}w_0w_1$ (word trigram)
surrounding POS tags	$p_{-2}p_{-1}p_0p_1p_2$ (POS 5-gram)
combinations	$w_{-1}w_0, w_0w_1, p_{-1}w_0, p_0w_0,$ $p_1w_0, p_0p_1p_2p_3, p_{-2}p_{-1}p_0,$ $p_{-1}p_0p_1, p_0p_1p_2, p_{-2}p_{-1},$ $p_{-1}p_0, p_0p_1, p_1p_2$

Johnson and Riezler (2000) suggested the possibility of the method for adapting a stochastic unification-based grammar including HPSG to another domain. They incorporated auxiliary distributions as additional features for an original log-linear model, and then attempted to assign proper weights to the new features. With this approach, they succeeded in decreasing to a degree indistinguishable sentences for a target grammar.

Our previous work proposed a method for adapting an HPSG parser trained on the Penn Treebank to a biomedical domain (Hara et al., 2005). We re-trained a disambiguation model of tree construction, i.e., q_{syn}, for the target domain. In this approach, q_{syn} of the original parser was used as a *reference distribution* (Jelinek, 1998) of another log-linear model, and the new model was trained using a target treebank. Since re-training used only a small treebank of the target domain, the cost was small and parsing accuracy was successfully improved.

3 Re-training of a Disambiguation Model of Lexical Entry Assignments

Our idea of domain adaptation is to train a disambiguation model of lexical entry assignments for the target domain and then incorporate it into the original parser. Since Enju includes the disambiguation model of lexical entry assignments as p_{lex}, we can implement our method in Enju by training another disambiguation model $p'_{lex}(l_i|\mathbf{w}, i)$ of lexical entry assignments for the biomedical domain, and then replacing the original p_{lex} with the newly trained p'_{lex}.

In this paper, for p'_{lex}, we train a disambiguation model $p_{lex-mix}(l_i|\mathbf{w}, i)$ of lexical entry assignments. $p_{lex-mix}$ is a maximum entropy model and the feature functions for it is the same as p_{lex} as given in Table 1. With these feature functions, we train $p_{lex-mix}$ on the treebanks both of the original and biomedical domains.

In the experiments, we examine the contribution of our method to parsing accuracy. In addition, we implement several other possible methods for comparison of the performances.

baseline: use the original model of Enju

GENIA only: execute the same method of training the disambiguation model of Enju, using only the GENIA treebank

Mixture: execute the same method of training the disambiguation model of Enju, using both of the Penn Treebank and the GENIA treebank (a kind of smoothing method)

HMT05: execute the method proposed in our previous work (Hara et al., 2005)

Our method: replace p_{lex} in the original model with $p_{lex-mix}$, while leaving q_{syn} as it is

Our method (GENIA): replace p_{lex} in the original model with $p_{lex-genia}$, which is a probabilistic model of lexical entry assignments trained only with the GENIA treebank, while leaving q_{syn} as it is

Our method + GENIA: replace p_{lex} in the original model with $p_{lex-mix}$ and q_{syn} with $q_{syn-genia}$, which is a disambiguation model of tree construction trained with the GENIA treebank

Our method + HMT05: replace p_{lex} in the original model with $p_{lex-mix}$ and q_{syn} with the model re-trained with our previous method (Hara et al., 2005) (the combination of our method and the "HMT05" method)

baseline (lex): use only p_{lex} as a disambiguation model

GENIA only (lex): use only $p_{lex-genia}$ as a disambiguation model, which is a probabilistic model of lexical entry assignments trained only with the GENIA treebank

Mixture (lex): use only $p_{lex-mix}$ as a disambiguation model

The "baseline" method does no adaptation to the biomedical domain, and therefore gives lower parsing accuracy for the domain than for the original domain. This method is regarded as the baseline of the experiments. The "GENIA only" method relies solely on the treebank for the biomedical domain, and therefore it cannot work well with the small treebank. The "Mixture" method is a kind of smoothing method using all available training data at the same time, and therefore the method can give the highest accuracy of the three, which would be regarded as the ideal accuracy with the naive methods. However, training this model is expected to be very costly.

The "baseline (lex)," "GENIA only (lex)," and "Mixture (lex)" approaches rely solely on models of lexical entry assignments, and show lower accuracy than those that contain both of models of lexical entry assignments and tree constructions. These approaches can be utilized as indicators of importance of combining the two types of models.

Our previous work (Hara et al., 2005) showed that the model trained with the "HMT05" method can give higher accuracy than the "baseline" method, even with the small amount of the treebanks in the biomedical domain. The model also takes much less cost to train than with the "Mixture" method. However, they reported that the method could not give as high accuracy as the "Mixture" method.

4 Experiments with the GENIA Corpus

4.1 Experimental Settings

We implemented the models shown in Section 3, and then evaluated the performance of them. The original parser, Enju, was developed on Section 02-21 of the Penn Treebank (39,832 sentences) (Miyao and Tsujii, 2005; Ninomiya et al., 2006). For training those models, we used the GENIA treebank (Kim et al., 2003), which consisted of 1,200 abstracts (10,848 sentences) extracted from MEDLINE. We divided it into three sets of 900, 150, and 150 abstracts (8,127, 1,361, and 1,360 sentences), and these sets were used respectively as training, development, and final evaluation data. The method of Gaussian MAP estimation (Chen and Rosenfeld, 1999) was used for smoothing. The meta parameter σ of the Gaussian distribution was determined so as to maximize the accuracy on the development set.

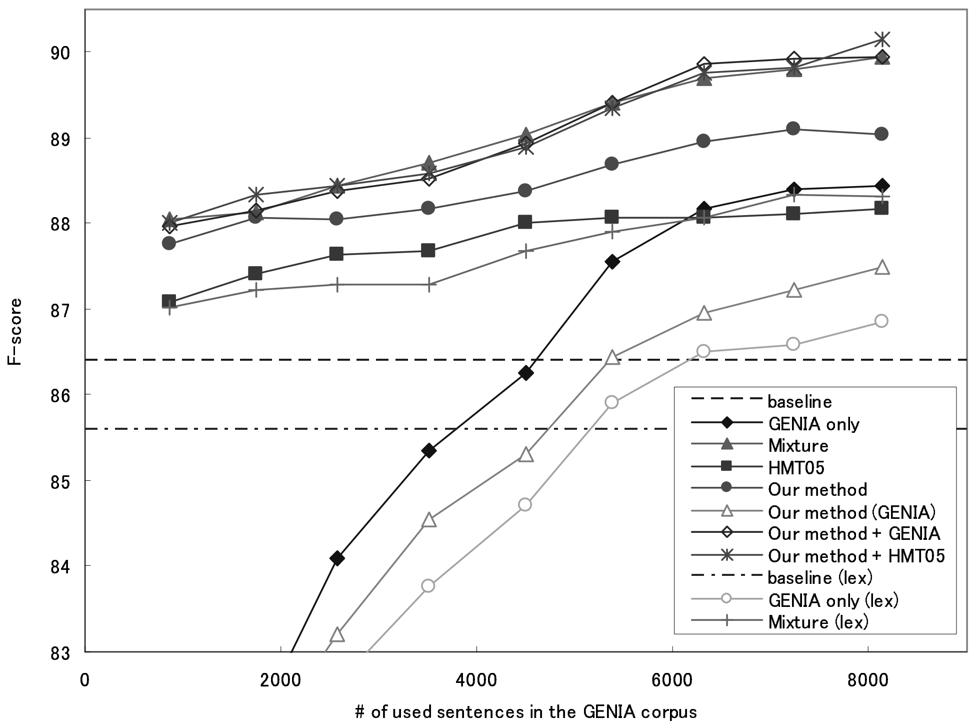


Figure 3: Corpus size vs. accuracy for various methods

In the following experiments, we measured the accuracy of predicate-argument dependencies on the evaluation set. The measure is labeled precision/recall (LP/LR), which is the same measure as previous work (Clark and Curran, 2004b; Miyao and Tsujii, 2005) that evaluated the accuracy of lexicalized grammars on the Penn Treebank.

The features for the examined approaches were all the same as the original disambiguation model. In our previous work, the features for "HMT05" were tuned to some extent. We evened out the features in order to compare various approaches under the same condition. The lexical entries for training each model were extracted from the treebank used for training the model of lexical entry assignments.

We compared the performances of the given models from various angles, by focusing mainly on the accuracy against the cost. For each of the models, we measured the accuracy transition according to the size of the GENIA treebank for training and according to the training time. We changed the size of the GENIA treebank for training: 100, 200, 300, 400, 500, 600, 700, 800, and 900 abstracts. Figure 3 and 4 show the F-score transition according to the size of the training set and the training time among the given models respectively. Table 2 and Table 3 show the parsing performance and the training cost obtained when using 900 abstracts of the GENIA treebank. Note that Figure 4 does not include the results of the "Mixture" method because only the method took too much training cost as shown in Table 3. It should also be noted that training time in Figure 4 includes time required for both training and development tests. In Table 2, accuracies with models other than "baseline" showed the significant differences from "baseline" according to stratified shuffling test (Cohen, 1995) with p-value < 0.05.

In the rest of this section we analyze these experimental results by focusing mainly on the contribution of re-training lexical entry assignment models. We first observe the results with the naive or existing approaches. On the basis of these results, we evaluate the impact of our method. We then explore the combination of our method with other methods, and analyze the errors for our future research.

4.2 Exploring Naive or Existing Approaches

Without adaptation, Enju gave the parsing accuracy of 86.39 in F-score, which was 3.42 point lower than

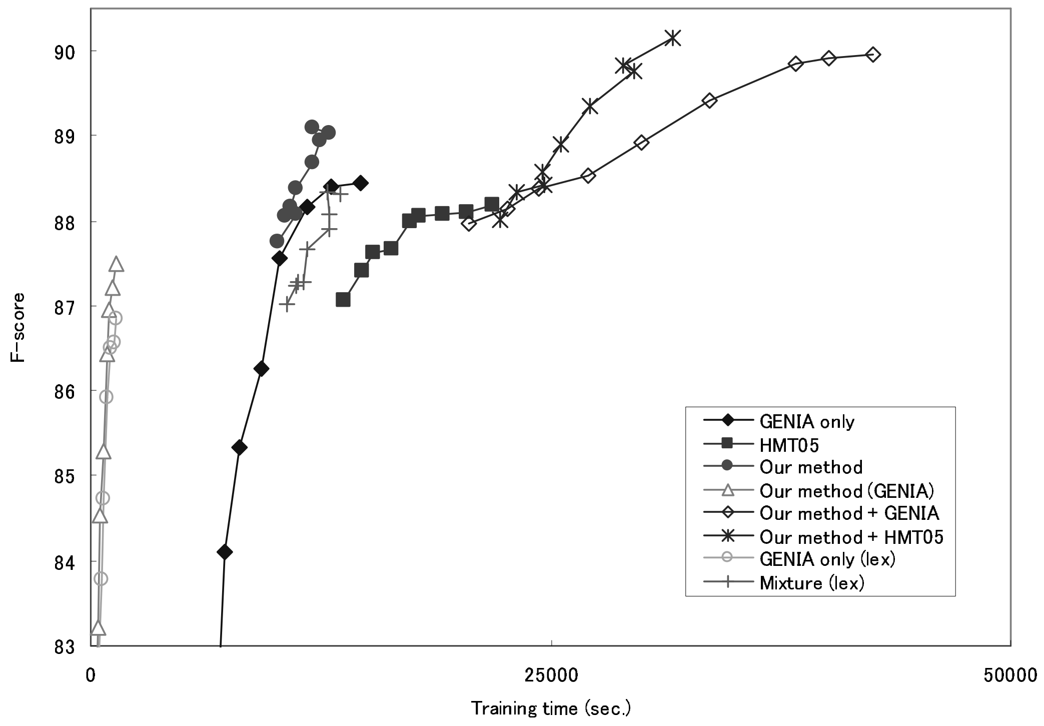


Figure 4: Training time vs. accuracy for various methods

that Enju gave for the original domain, the Penn Treebank. This is the baseline of the experiments.

Figure 3 shows that, for less than about 4,500 training sentences, the “GENIA only” method could not obtain as high parsing accuracy as the “baseline” method. This result would indicate that the training data would not be sufficient for re-training the whole disambiguation model from scratch. However, if we prepared more than about 4,500 sentences, the method could give higher accuracy than “baseline” with low training cost (see Figure 4). On the other hand, the “Mixture” method could obtain the highest level of the parsing accuracy for any size of the GENIA treebank. However, Table 3 shows that this method required too much training cost. It would be a major barrier for further challenges for improvement with various additional parameters.

The “HMT05” method could give higher accuracy than the “baseline” method for any size of the training sentences although the accuracy was lower than the “Mixture” method. The method could also be carried out in much smaller training time and lower cost than the “Mixture” method. These points would be the benefits of the “HMT05” method. On the other hand, when we compared the “HMT05” method with the “GENIA only” method, for the larger size of the training corpus, the “HMT05” method was defeated by the “GENIA only” method in parsing accuracy and training cost.

4.3 Impact of Re-training a Lexical Disambiguation Model

When we focused on our method, it could constantly give higher accuracy than the “baseline” and the “HMT05” methods. These results would indicate that, for an individual method, re-training a model of lexical entry assignments might be more critical to domain adaptation than re-training that of tree construction. In addition, for the small treebank, our method could give as high accuracy as the “Mixture” method with much lower training cost. Our method would be a very satisfiable approach when applied with a small treebank. It should be noted that the re-trained lexical model could not solely give the accuracy as high as our method (see “Mixture (lex)” in Figure 3). The combination of a re-trained lexical model and a tree construction model would have given such a high performance.

When we compared the training time for our

Table 2: Parsing accuracy and time for various methods

	For GENIA Corpus				For Penn Treebank			
	LP	LR	F-score	Time	LP	LR	F-score	Time
baseline	86.71	86.08	86.39	476 sec.	89.99	89.63	89.81	675 sec.
GENIA only	88.99	87.91	88.45	242 sec.	72.07	45.78	55.99	2,441 sec.
Mixture	90.01	89.87	89.94	355 sec.	89.93	89.60	89.77	767 sec.
HMT05	88.47	87.89	88.18	510 sec.	88.92	88.61	88.76	778 sec.
Our method	89.11	88.97	89.04	327 sec.	89.96	89.63	89.79	713 sec.
Our method (GENIA)	86.06	85.15	85.60	542 sec.	70.18	44.88	54.75	3,290 sec.
Our method + GENIA	90.02	89.88	89.95	320 sec.	88.11	87.77	87.94	718 sec.
Our method + HMT05	90.23	90.08	90.15	377 sec.	89.31	88.98	89.14	859 sec.
baseline (lex)	85.93	85.27	85.60	377 sec.	87.52	87.13	87.33	553 sec.
GENIA only (lex)	87.42	86.28	86.85	197 sec.	71.49	45.41	55.54	1,928 sec.
Mixture (lex)	88.43	88.18	88.31	258 sec.	87.49	87.12	87.30	585 sec.

Table 3: Training cost of various methods

	Training time	Memory used
baseline	0 sec.	0.00 GByte
GENIA only	14,695 sec.	1.10 GByte
Mixture	238,576 sec.	5.05 GByte
HMT05	21,833 sec.	1.10 GByte
Our method	12,957 sec.	4.27 GByte
Our method (GENIA)	1,419 sec.	0.94 GByte
Our method + GENIA	42,475 sec.	4.27 GByte
Our method + HMT05	31,637 sec.	4.27 GByte
baseline (lex)	0 sec.	0.00 GByte
GENIA only (lex)	1,434 sec.	1.10 GByte
Mixture (lex)	13,595 sec.	4.27 GByte

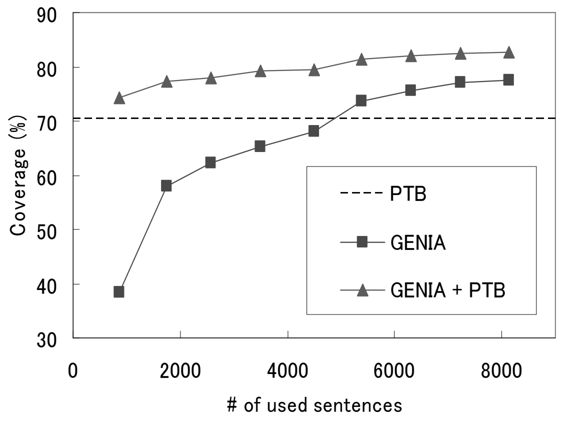


Figure 5: Corpus size vs. coverage of each training set for the GENIA corpus

method with the one for the "HMT05" method, our method required less time than the "HMT05" method. This would be because our method required only the re-training of the very simple model, that is, a probabilistic model of lexical entry assignments.

It should be noted that our method would not work only with in-domain treebank. The "Our method (GENIA)" and the "GENIA only (lex)" methods could hardly give as high parsing accuracy as the "baseline" method. Although, for the larger size of the GENIA treebank, the methods could obtain a little higher accuracy than the "baseline" method, the benefit was very little. These results would indicate that only the treebank in the target domain would be insufficient for adaptation. Figure 5 shows the coverage of each training corpus for the GENIA treebank, which would also support the above observation. It shows that the GENIA treebank could not solely cover so much sentences in the GENIA corpus as the combination of the Penn Treebank and the GENIA treebank.

Table 4: Coverage of each training set

Training set	% of covered sentences	
	for GENIA	for PTB
GENIA treebank	77.54 %	25.66 %
PTB treebank	70.45 %	84.12 %
GENIA treebank + PTB treebank	82.74 %	84.86 %

4.4 Effectiveness of Combining Lexical and Syntactic Disambiguation Models

When we compared the "Our method + HMT05" and "Our method + GENIA" methods with the "Mixture" method, the former two models could give as the high parsing accuracies as the latter one for any size of the training corpus. In particular, for the maximum size, the "Our method + HMT05" models could give a little higher parsing accuracy than the "Mixture" method. This difference was

Table 5: Errors in various methods

	Total errors	=	Common errors with baseline	+	Specific errors
GENIA only	2,889	=	1,906 (65.97%)	+	983 (34.03%)
Mixture	2,653	=	2,177 (82.06%)	+	476 (17.94%)
HMT05	3,063	=	2,470 (80.64%)	+	593 (19.36%)
Our method	2,891	=	2,405 (83.19%)	+	486 (16.81%)
Our method (GENIA)	3,153	=	2,070 (65.65%)	+	1,083 (34.35%)
Our method + GENIA	2,650	=	2,056 (77.58%)	+	594 (22.42%)
Our method + HMT05	2,597	=	1,943 (74.82%)	+	654 (25.18%)
baseline	3,542				
	Total errors	=	Common errors with baseline (lex)	+	Specific errors
GENIA only (lex)	3,320	=	2,509 (75.57%)	+	811 (24.43%)
Mixture (lex)	3,100	=	2,769 (89.32%)	+	331 (10.68%)
baseline (lex)	3,757				

Table 6: Types of disambiguation errors

Error cause	# of errors		
	Common	Only for	
		Baseline	Adapted
Attachment ambiguity			
prepositional phrase	12	12	6
relative clause	0	1	0
adjective	4	2	2
adverb	1	3	1
verb phrase	10	3	1
subordinate clause	0	2	0
Argument/modifier distinction			
to-infinitive	0	0	7
Lexical ambiguity			
preposition/modifier	0	3	0
verb subcategorization frame	5	0	6
participle/adjective	0	2	0
Test set errors			
Errors of treebank	2	0	0
Other types of error causes			
Comma	10	8	4
Noun phrase identification	21	5	8
Coordination/insertion	6	3	5
Zero-pronoun resolution	8	1	0
Others	1	1	2

shown to be significant according to stratified shuffling test with p-value < 0.10, which might suggest the beneficial impact of the "Our method + HMT05" method. In addition, Figure 4 and Table 3 show that training the "Our method + HMT05" or "Our method + GENIA" model required much less time and PC memory than training the "Mixture" model. According to the above observation, we would be able to say that the "Our method + HMT05" method might be the most ideal among the given methods.

The "Our method + HMT05" and "Our method + GENIA" methods showed the different performances in the point that the former could obtain high parsing accuracy with less training time than the latter. This would come from the fact that the latter method trained $q_{syn-genia}$ solely with lexical entries in the GENIA treebank, while the former one trained q_{syn} with rich lexical entries borrowed from $q_{lex-mix}$. Rich lexical entries would decrease unknown lexical entries, and therefore would improve the effectiveness of making the feature forest model. On the other hand, the difference in lexical entries would not seem to affect so much on the contribution of tree construction model to the parsing accuracy. In our experiments, the parameters for a tree construction model such as feature functions were not adjusted thoroughly, which might possibly blur the benefits of the rich lexical entries.

4.5 Error Analysis

Table 5 shows the comparison of the number of errors for various models with that for the original model in parsing the GENIA corpus. For each of the methods, the table gives the numbers of common errors with the original Enju model and the ones specific to that method. If possible, we would like our methods to decrease the errors in the original Enju model while not increasing new errors. The table shows that our method gave the least percentage of newly added errors among the approaches except for the methods utilizing only lexical entry assignments models. On the other hand, the "Our method + HMT05" approach gave over 25 % of newly added errors, although we considered above that the approach gave the best performance.

In order to explore this phenomenon, we observed the errors for the "Our method + HMT05" and the baseline models, and then classified them into several types. Table 6 shows manual classification of causes of errors for the two models in 50 sentences. In the classification, one error often propagated and resulted in multiple errors of predicate argument dependencies. The numbers in the table include such double counting. It would be desirable that the errors in the rightmost column were less than the ones in the middle column, which means that the "Our method + HMT05" approach did not produce more errors specific to the approach than the baseline.

With the "Our method + HMT05" approach, errors for "attachment ambiguity" decreased as a whole. Errors for "comma" and lexical ambiguities of "preposition/modifier" and "participle/adjective" also decreased. For these attributes, the approach could learn in the training phase lexical properties of continuous words with the lexical entry assignment model, and syntactic relations of separated words with the tree construction model. On the other hand, the errors for "to-infinitive argument/modifier distinction" and "verb subcategorization frame ambiguity" considerably increased. These two types of errors have close relation to each other because the failure to recognize verb subcategorization frames tends to cause the failure to recognize the syntactic role of the to-infinitives. We must research further on these errors in our future work.

When we focused on "noun phrase identification," most of the errors did not differ between the two models. In the biomedical domain, there would be many technical terms which could not be correctly identified solely with the disambiguation model, which would possibly result in such many untouched errors. In order to properly cope with these terms, we might have to introduce some kinds of dictionaries or named entity recognition methods.

5 Experiments with the Brown Corpus

5.1 Brown Corpus

We applied our methods to the Brown corpus (Kucera and Francis, 1967) and examined the portability of our method. The Brown corpus consists of 15 domains, and the Penn Treebank gives bracketed version of the corpus for the 8 domains containing 19,395 sentences (Table 7).

Table 7: Domains in the Brown corpus

label	domain	sentences
CF	popular lore	2,420
CG	belles lettres	2,546
CK	general fiction	3,172
CL	mystery and detective fiction	2,745
CM	science fiction	615
CN	adventure and western fiction	3,521
CP	romance and love story	3,089
CR	humor	812
All	total of all the above domains	19,395

For the target of adaptation, we utilized the domain containing all of these 8 domains as a total fiction domain (labelled "All") as well as the individual ones. As in the experiments with the GENIA Treebank, we divided sentences for each domain into three parts, 80% for training, 10% for develepment test, and 10% for final test. For the "All" domain, we merged all training sets, all development test sets, and all final test sets for the 8 domains respectively.

Table 8 and 9 show the parsing accuracy and training time for each domain with the various methods shown in Section 3. The methods are fundamentally the same as in the experiments with the GENIA corpus except that the target corpus is replaced with the Brown corpus. In order to avoid confusion, we replaced "GENIA" in the names of the methods with "Brown." Each of the bold numbers in Table 8 means that it was the best accuracy given for the target domain. It should be noted that the "CM" and "CR" domain contains very small treebank, and therefore we must consider that the results with these domains would not be so useful.

5.2 Evaluation of Portability of Our Method

When we focus on the "ALL" domain, the approaches other than the baseline succeeded to give higher parsing accuracy than the baseline. This would show that these approaches were effective not only for the GENIA corpus but also for the Brown corpus. The "Mixture" method gave the highest accuracy which was 3.41 point higher than the baseline. The "Our method + HMT05" approach also gave the accuracy as high as the "Mixture" method. In addition, as is the case with the GENIA corpus, the approach could be trained with much less time than the "Mixture" method. Not only for these two

Table 8: Parsing accuracy for the Brown corpus

	F-score								
	ALL	**CF**	**CG**	**CK**	**CL**	**CM**	**CN**	**CP**	**CR**
baseline	83.09	85.75	85.38	81.12	77.53	85.30	82.84	85.18	76.63
Brown only	84.84	77.65	78.92	75.72	70.56	50.02	78.38	79.10	50.34
Mixture	**86.50**	86.59	**85.94**	82.49	**78.66**	84.82	84.28	86.85	76.45
HMT05	83.79	85.80	84.98	81.48	76.91	85.25	83.50	85.66	77.15
Our method	86.14	86.73	85.74	82.77	77.95	85.40	84.23	**86.90**	76.71
Our method (GENIA)	84.71	78.49	79.63	75.43	70.86	50.24	78.49	79.69	51.82
Our method + GENIA	86.00	86.12	85.41	**83.22**	77.10	83.39	84.21	85.77	76.91
Our method + HMT05	86.44	**86.76**	85.85	82.90	77.70	**85.61**	**84.43**	86.87	77.48
baseline (lex)	82.19	84.69	83.85	80.25	76.32	83.42	81.29	84.13	77.33
Brown only (lex)	83.92	77.12	77.81	75.06	70.35	49.95	77.06	78.84	50.63
Mixture (lex)	85.29	85.47	84.18	81.88	77.22	83.98	82.67	85.65	**77.58**

Table 9: Consumed time for various methods for the Brown corpus

	Consumed time for training (sec.)								
	ALL	**CF**	**CG**	**CK**	**CL**	**CM**	**CN**	**CP**	**CR**
baseline	0	0	0	0	0	0	0	0	0
Brown only	42,614	4,115	3,763	2,478	2,162	925	2,362	2,695	1,226
Mixture	383,557	190,449	159,490	156,299	210,357	131,335	170,108	224,045	184,251
HMT05	30,933	6,003	4,830	4,186	5,010	1,681	4,411	5,069	1,588
Our method	15,912	11,053	10,988	11,151	10,782	10,158	11,075	10,594	10,284
Our method (Brown)	3,273	312	373	310	249	46	321	317	86
Our method + Brown	130,434	24,633	21,848	20,171	19,184	11,995	19,164	20,922	13,461
Our method + HMT05	54,355	17,722	16,627	15,229	14,914	12,226	15,760	16,175	11,724
baseline (lex)	0	0	0	0	0	0	0	0	0
Brown only (lex)	3,001	317	373	308	251	47	321	317	86
Mixture (lex)	21,148	11,128	11,251	11,094	10,728	10,466	11,151	10,897	10,537

methods, the experimental results for the "All" domain showed the tendency similar to the GENIA corpus as a whole, except for the less improvement with the "HMT05" method.

When we focus on the individual domains, our method could successfully obtain higher parsing accuracy than the baseline for all the domains. Moreover, for the "CP" domain, our method could give the highest parsing accuracy among the methods. These results would support the portability of retraining the model for lexical entry assignment. The "Our method + HMT05" approach, which gave the highest performance for the GENIA corpus, also gave accuracy improvement for the all domains while it did not give so much impact for the "CL" domain. The "Mixture" approach, which utilized the same lexical entry assignment model, could obtain 0.94 point higher parsing accuracy than the "Our method + HMT05" approach. Table 10, which shows the lexical coverage with each domains, does not seem to indicate the noteworthy difference in lexical entry coverage between the "CL" and the other domains. As mentioned in the error analysis in Section 4, the model of tree construction might affect the performance in some way. In our future work, we must clarify the mechanism of this result and would like to further improve the performance.

6 Related Work

For recent years, domain adaptation has been studied extensively. This section explores how our research is relevant to the previous works.

Our previous work (Hara et al., 2005) and this research mainly focused on how to draw as much benefit from a smaller amount of in-domain annotated data as possible. Titov and Henderson (2006) also took this type of approach. They first trained a probabilistic model on original and target treebanks and used it to define a kernel over parse trees. This kernel was used in a large margin classifier trained on a small set of data only from the target domain, and the classifier was then used for reranking the top

Table 10: Coverage of each training set for the Brown corpus

Training set	% of covered sentences for the target corpus								
	ALL	**CF**	**CG**	**CK**	**CL**	**CM**	**CN**	**CP**	**CR**
Target treebank	74.99 %	49.13 %	50.00 %	47.97 %	49.08 %	29.66 %	53.51 %	64.01 %	8.57%
PTB treebank	70.02 %	72.09 %	68.93 %	66.42 %	68.87 %	78.62 %	70.00 %	77.59 %	47.14 %
Target + PTB	79.77 %	74.71 %	71.47 %	71.59 %	70.45 %	80.00 %	72.70 %	80.39 %	52.86 %

parses on the target domain.

On the other hand, several studies have explored how to draw useful information from unlabelled in-domain data. Roark and Bacchiani (2003) adapted a lexicalized PCFG by using maximum *a posteriori* (MAP) estimation for handling unlabelled adaptation data. In the field of classifications, Blitzer et al. (2006) utilized unlabelled corpora to extract features of structural correspondences, and then adapted a POS-tagger to a biomedical domain. Steedman et al. (2003) utilized a co-training parser for adaptation and showed that co-training is effective even across domains. McClosky et al. (2006) adapted a re-ranking parser to a target domain by self-training the parser with unlabelled data in the target domain. Clegg and Shepherd (2005) combined several existing parsers with voting schemes or parse selection, and then succeeded to gain the improvement of performance for a biomedical domain. Although unsupervised methods can exploit large in-domain data, the above studies could not obtain the accuracy as high as that for an original domain, even with the sufficient size of the unlabelled corpora. On the other hand, we showed that our approach could achieve this goal with about 6,500 labelled sentences. However, when 6,500 labelled can not be prepared, it might be worth while to explore the potentiality of combining the above unsupervised and our supervised methods.

When we focuses on biomedical domains, there have also been various works which coped with domain adaptation. Biomedical sentences contain many technical terms which cannot be easily recognized without expert knowledge, and this damages performances of NLP tools directly. In order to solve this problem, two types of approaches have been suggested. The first approach is to utilize existing domain-specific lexical resources. Lease and Charniak (2005) utilized POS tags, dictionary collocations, and named entities for parser adaptation, and then succeeded to achieve accuracy improvement. The second approach is to expand lexical entries for a target domain. Szolovits (2003) extended a lexical dictionary for a target domain by predicting lexical information for words. They transplanted lexical *indiscernibility* of words in an original domain into a target domain. Pyysalo et al. (2004) showed the experimental results that this approach improved the performance of a parser for Link Grammar. Since our re-trained model of lexical entry assignments was shown to be unable to cope with this problem properly (shown in Section 4), the combination of the above approaches with our approach would be expected to bring further improvement.

7 Conclusions

This paper presented an effective approach to adapting an HPSG parser trained on the Penn Treebank to a biomedical domain. We trained a probabilistic model of lexical entry assignments in a target domain and then incorporated it into the original parser. The experimental results showed that this approach obtains higher parsing accuracy than the existing approach of adapting the structural model alone. Moreover, the results showed that, the combination of our method and the existing approach could achieve parsing accuracy that is as high as that obtained by re-training an HPSG parser for the target domain from scratch, but with much lower training cost. With this model, the parsing accuracy for the target domain improved by 3.84 f-score points, using a domain-specific treebank of 8,127 sentences. Experiments showed that 6,500 sentences are sufficient for achieving as high parsing accuracy as the baseline for the original domain.

In addition, we applied our method to the Brown corpus in order to evaluate the portability of our method. Experimental results showed that the parsing accuracy for the target domain improved by 3.35 f-score points. On the other hand, when we focused on some individual domains, that combination approach could not give the desirable results.

In future work, we would like to explore further performance improvement of our approach. For the first step, domain-specific features such as named entities could be much help for solving unsuccessful recognition of technical terms.

Acknowledgment

This research was partially supported by Grant-in-Aid for Specially Promoted Research 18002007.

References

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2).

A. L. Berger, S. A. Della Pietra, and V. J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

J. Blitzer, R. McDonald, and F. Pereira. 2006. Domain adaptation with structural correspondence learning. In *Proc. EMNLP 2006*.

Y. S. Chan and H. T. Ng. 2006. Estimating class priors in domain adaptation for word sense disambiguation. In *Proc. 21st COLING and 44th ACL*.

S. Chen and R. Rosenfeld. 1999. A gaussian prior for smoothing maximum entropy models. Technical Report CMUCS-99-108, Carnegie Mellon University.

S. Clark and J. R. Curran. 2004a. The importance of supertagging for wide-coverage CCG parsing. In *Proc. COLING-04*.

S. Clark and J. R. Curran. 2004b. Parsing the WSJ using CCG and log-linear models. In *Proc. 42nd ACL*.

S. Clark and J. R. Curran. 2006. Partial training for a lexicalized-grammar parser. In *Proc. NAACL-06*.

A. B. Clegg and A. Shepherd. 2005. Evaluating and integrating treebank parsers on a biomedical corpus. In *Proc. the ACL Workshop on Software*.

P. R. Cohen. 1995. *Empirical Methods for Artificial Intelligence*. MIT Press.

T. Hara, Y. Miyao, and J. Tsujii. 2005. Adapting a probabilistic disambiguation model of an HPSG parser to a new domain. In *Proc. IJCNLP 2005*.

F. Jelinek. 1998. *Statistical Methods for Speech Recognition*. The MIT Press.

M. Johnson and S. Riezler. 2000. Exploiting auxiliary distributions in stochastic unification-based grammars. In *Proc. 1st NAACL*.

J. D. Kim, T. Ohta, Y. Teteisi, and J. Tsujii. 2003. GENIA corpus - a semantically annotated corpus for biotextmining. *Bioinformatics*, 19(suppl. 1):i180–i182.

H. Kucera and W. N. Francis. 1967. *Computational Analysis of Present-Day American English*. Brown University Press, Providence, RI.

M. Lease and E. Charniak. 2005. Parsing biomedical literature. In *In Proc. IJCNLP 2005*.

M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. 1994. The Penn Treebank: Annotating predicate argument structure. In *ARPA HLT Workshop*.

D. McClosky, E. Charniak, and M. Johnson. 2006. Reranking and self-training for parser adaptation. In *Proc. 21st COLING and 44th ACL*.

Y. Miyao and J. Tsujii. 2002. Maximum entropy estimation for feature forests. In *Proc. HLT 2002*.

Y. Miyao and J. Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proc. ACL 2005*.

Y. Miyao, T. Ninomiya, and J. Tsujii. 2004. Corpus-oriented grammar development for acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank. In *Proc. IJCNLP-04*.

T. Ninomiya, T. Matsuzaki, Y. Tsuruoka, Y. Miyao, and J. Tsujii. 2006. Extremely lexicalized models for accurate and fast HPSG parsing. In *Proc. EMNLP 2006*.

C. Pollard and I. A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

S. Pyysalo, F. Ginter, T. Pahikkala, J. Koivula, J. Boberg, J. Jrvinen, and T. Salakoski. 2004. Analysis of Link Grammar on biomedical dependency corpus targeted at protein-protein interactions. In *Proc. BioNLP/NLPBA 2004*.

B. Roark and M. Bacchiani. 2003. Supervised and unsupervised PCFG adaptation to novel domains. In *Proc. HLT-NAACL 2003*.

M. Steedman, M. Osborne, A. Sarkar, S. Clark, R. Hwa, J. Hockenmaier, P. Ruhen, S. Baker, and J. Crim. 2003. Bootstrapping statistical parsers from small datasets. In *Proc. European ACL (EACL)*.

P. Szolovits. 2003. Adding a medical lexicon to an English parser. In *AMIA Annu Symp Proc*.

Ivan Titov and James Henderson. 2006. Porting statistical parsers with data-defined kernels. In *Proc. CoNLL-2006*.

Semi-supervised Training of a Statistical Parser from Unlabeled Partially-bracketed Data

Rebecca Watson and Ted Briscoe
Computer Laboratory
University of Cambridge, UK
FirstName.LastName@cl.cam.ac.uk

John Carroll
Department of Informatics
University of Sussex, UK
J.A.Carroll@sussex.ac.uk

Abstract

We compare the accuracy of a statistical parse ranking model trained from a fully-annotated portion of the Susanne treebank with one trained from unlabeled partially-bracketed sentences derived from this treebank and from the Penn Treebank. We demonstrate that *confidence-based* semi-supervised techniques similar to self-training outperform expectation maximization when both are constrained by partial bracketing. Both methods based on partially-bracketed training data outperform the fully supervised technique, and both can, in principle, be applied to any statistical parser whose output is consistent with such partial-bracketing. We also explore tuning the model to a different domain and the effect of in-domain data in the semi-supervised training processes.

1 Introduction

Extant statistical parsers require extensive and detailed treebanks, as many of their lexical and structural parameters are estimated in a fully-supervised fashion from treebank derivations. Collins (1999) is a detailed exposition of one such ongoing line of research which utilizes the Wall Street Journal (WSJ) sections of the Penn Treebank (PTB). However, there are disadvantages to this approach. Firstly, treebanks are expensive to create manually. Secondly, the richer the annotation required, the harder it is to adapt the treebank to train parsers which make different assumptions about the structure of syntactic analyses. For example, Hockenmeier (2003) trains a statistical parser based on Combinatory Categorial Grammar (CCG) on the WSJ PTB, but first maps the treebank to CCG derivations semi-automatically. Thirdly, many (lexical) parameter estimates do not generalize well between domains. For instance, Gildea (2001) reports that WSJ-derived bilexical parameters in Collins' (1999) Model 1 parser contribute about 1% to parse selection accuracy when test data is in the same domain, but yield no improvement for test data selected from the Brown Corpus. Tadayoshi *et al.* (2005) adapt a statistical parser trained on the WSJ PTB to the biomedical domain by retraining on the Genia Corpus, augmented with manually corrected derivations in the same format. To make statistical parsing more viable for a range of applications, we need to make more effective and flexible use of extant training data and minimize the cost of annotation for new data created to tune a system to a new domain.

Unsupervised methods for training parsers have been relatively unsuccessful to date, including expectation maximization (EM) such as the inside-outside algorithm (IOA) over PCFGs (Baker, 1979; Prescher, 2001). However, Pereira and Schabes (1992) adapted the IOA to apply over semi-supervised data (unlabeled bracketings) extracted from the PTB. They constrain the training data (parses) considered within the IOA to those consistent with the constituent boundaries defined by the bracketing. One advantage of this approach is that, although less information is derived from the treebank, it generalizes better to parsers which make different representational assumptions, and it is easier, as Pereira and Schabes did, to map unlabeled bracketings to a format more consistent with the target grammar. Another is that the cost of annotation with unlabeled brackets should be lower than that of developing a representationally richer treebank. More recently, both Riezler *et al.* (2002) and Clark and Curran (2004) have successfully trained maximum entropy parsing models utilizing all derivations in the model consistent with the annotation of the WSJ PTB, weighting counts by the normalized probability of the associated derivation. In this paper, we extend this line of investigation by utilizing only unlabeled and partial bracketing.

We compare the performance of a statistical parsing model trained from a detailed treebank with that of the same model trained with semi-supervised techniques that require only unlabeled partially-bracketed data. We contrast an IOA-based EM method for training a PGLR parser (Inui *et al.*, 1997), similar to the method applied by Pereira and Schabes to PCFGs, to a range of *confidence-based* semi-supervised methods described below. The IOA is a generalization of the Baum-Welch or Forward-Backward algorithm, another instance of EM, which can be used to train Hidden Markov Models (HMMs). Elworthy (1994) and Merialdo (1994) demonstrated that Baum-Welch does not necessarily improve the performance of an HMM part-of-speech tagger when deployed in an unsupervised or semi-supervised setting. These somewhat negative results, in contrast to those of Pereira and Schabes (1992), suggest that EM techniques require fairly determinate training data to yield useful models. Another motivation to explore alternative non-iterative methods is that the derivation space over partially-bracketed data can remain large (>1K) while the *confidence-based* methods we explore have a total processing overhead equivalent to one iteration of an IOA-based EM algorithm.

As we utilize an initial model to annotate additional training data, our methods are closely related to self-training methods described in the literature (e.g. McClosky *et al.* 2006, Bacchiani *et al.* 2006). However these methods have been applied to fully-annotated training data to create the initial model, which is then used to annotate further training data derived from unannotated text. Instead, we train entirely from partially-bracketed data, starting from the small proportion of 'unambiguous' data whereby a single parse is consistent with the annotation. Therefore, our methods are better described as semi-supervised and the main focus of this work is the flexible re-use of existing treebanks to train a wider variety of statistical parsing models. While many statistical parsers extract a context-free grammar in parallel with a statistical parse selection model, we demonstrate that existing treebanks can be utilized to train parsers that deploy grammars that make other representational assumptions. As a result, our methods can be applied by a range of parsers to minimize the manual effort required to train a parser or adapt to a new domain.

§2 gives details of the parsing system that are relevant to this work. §3 and §4 describe our data and evaluation schemes, respectively. §5 describes our semi-supervised training methods. §6 explores the problem of tuning a parser to a new domain. Finally, §7 gives conclusions and future work.

2 The Parsing System

Sentences are automatically preprocessed in a series of modular pipelined steps, including tokenization, part of speech (POS) tagging, and morphological analysis, before being passed to the statistical parser. The parser utilizes a manually written feature-based unification grammar over POS tag sequences.

2.1 The Parse Selection Model

A context-free 'backbone' is automatically derived from the unification grammar[1] and a generalized or non-deterministic LALR(1) table is

[1]This backbone is determined by compiling out the values of prespecified attributes. For example, if we compile out the attribute PLURAL which has 2 possible values (plural or not) we will create 2 CFG rules for each rule with categories that contain PLURAL. Therefore, no information is lost during this process.

constructed from this backbone (Tomita, 1987). The residue of features not incorporated into the backbone are unified on each reduce action and if unification fails the associated derivation paths also fail. The parser creates a packed parse forest represented as a graph-structured stack.[2] The parse selection model ranks complete derivations in the parse forest by computing the product of the probabilities of the (shift/reduce) parse actions (given LR state and lookahead item) which created each derivation (Inui *et al.*, 1997).

Estimating action probabilities, consists of a) recording an action history for the correct derivation in the parse forest (for each sentence in a treebank), b) computing the frequency of each action over all action histories and c) normalizing these frequencies to determine probability distributions over conflicting (i.e. shift/reduce or reduce/reduce) actions.

Inui *et al.* (1997) describe the probability model utilized in the system where a transition is represented by the probability of moving from one stack state, σ_{i-1}, (an instance of the graph structured stack) to another, σ_i. They estimate this probability using the stack-top state s_{i-1}, next input symbol l_i and next action a_i. This probability is conditioned on the type of state s_{i-1}. S_s and S_r are mutually exclusive sets of states which represent those states reached after shift or reduce actions, respectively. The probability of an action is estimated as:

$$P(l_i, a_i, \sigma_i | \sigma_{i-1}) \approx \left\{ \begin{array}{ll} P(l_i, a_i | s_{i-1}) & s_{i-1} \in S_s \\ P(a_i | s_{i-1}, l_i) & s_{i-1} \in S_r \end{array} \right\}$$

Therefore, normalization is performed over all lookaheads for a state or over each lookahead for the state depending on whether the state is a member of S_s or S_r, respectively (hereafter the I function). In addition, Laplace estimation can be used to ensure that all actions in the table are assigned a non-zero probability (the I_L function).

[2]The parse forest is an instance of a *feature forest* as defined by Miyao and Tsujii (2002). We will use the term 'node' herein to refer to an element in a derivation tree or in the parse forest that corresponds to a (sub-)analysis whose label is the mother's label in the corresponding CF 'backbone' rule.

3 Training Data

The treebanks we use in this work are in one of two possible formats. In either case, a treebank T consists of a set of sentences. Each sentence t is a pair (s, M), where s is the automatically preprocessed set of POS tagged tokens (see §2) and M is either a fully annotated derivation, A, or an unlabeled bracketing U. This bracketing may be partial in the sense that it may be compatible with more than one derivation produced by a given parser. Although occasionally the bracketing is itself complete but alternative non-terminal labeling causes indeterminacy, most often the 'flatter' bracketing available from extant treebanks is compatible with several alternative 'deeper' mostly binary-branching derivations output by a parser.

3.1 Derivation Consistency

Given $t = (s, A)$, there will exist a single derivation in the parse forest that is compatible (correct). In this case, equality between the derivation tree and the treebank annotation A identifies the correct derivation. Following Pereira and Schabes (1992) given $t = (s, U)$, a node's span in the parse forest is *valid* if it does not overlap with any span outlined in U, and hence, a derivation is correct if the span of every node in the derivation is valid in U. That is, if no crossing brackets are present in the derivation. Thus, given $t = (s, U)$, there will often be more than one derivation compatible with the partial bracketing.

Given the correct nodes in the parse forest or in derivations, we can then extract the corresponding action histories and estimate action probabilities as described in §2.1. In this way, partial bracketing is used to constrain the set of derivations considered in training to those that are compatible with this bracketing.

3.2 The Susanne Treebank and Baseline Training Data

The Susanne Treebank (Sampson, 1995) is utilized to create fully annotated training data.

This treebank contains detailed syntactic derivations represented as trees, but the node labeling is incompatible with the system grammar. We extracted sentences from Susanne and automatically preprocessed them. A few multiwords are retokenized, and the sentences are retagged using the POS tagger, and the bracketing deterministically modified to more closely match that of the grammar, resulting in a bracketed corpus of 6674 sentences. We will refer to this bracketed treebank as S, henceforth.

A fully-annotated and system compatible treebank of 3543 sentences from S was also created. We will refer to this annotated treebank, used for fully supervised training, as B. The system parser was applied to construct a parse forest of analyses which are compatible with the bracketing. For 1258 sentences, the grammar writer interactively selected correct (sub)analyses within this set until a single analysis remained. The remaining 2285 sentences were automatically parsed and all consistent derivations were returned. Since B contains more than one possible derivation for roughly two thirds of the data the 1258 sentences (paired with a single tree) were repeated twice so that counts from these trees were weighted more highly. The level of reweighting was determined experimentally using some held out data from S. The baseline supervised model against which we compare in this work is defined by the function $I_L(B)$ as described in §2.1. The costs of deriving the fully-annotated treebank are high as interactive manual disambiguation takes an average of ten minutes per sentence, even given the partial bracketing derived from Susanne.

3.3 The WSJ PTB Training Data

The Wall Street Journal (WSJ) sections of the Penn Treebank (PTB) are employed as both training and test data by many researchers in the field of statistical parsing. The annotated corpus implicitly defines a grammar by providing a labeled bracketing over words annotated with POS tags. We extracted the unlabeled bracketing from the de facto standard training sections (2-21 inclusive).[3] We will refer to the resulting corpus as W and the combination (concatenation) of the partially-bracketed corpora S and W as SW.

3.4 The DepBank Test Data

King *et al.* (2003) describe the development of the PARC 700 Dependency Bank, a gold-standard set of relational dependencies for 700 sentences (from the PTB) drawn at random from section 23 of the WSJ (the de facto standard test set for statistical parsing). In all the evaluations reported in this paper we test our parser over a gold-standard set of relational dependencies compatible with our parser output derived (Briscoe and Carroll, 2006) from the PARC 700 Dependency Bank (DepBank, henceforth).

The Susanne Corpus is a (balanced) subset of the Brown Corpus which consists of 15 broad categories of American English texts. All but one category (reportage text) is drawn from different domains than the WSJ. We therefore, following Gildea (2001) and others, consider S, and also the baseline training data, B, as out-of-domain training data.

4 The Evaluation Scheme

The parser's output is evaluated using a relational dependency evaluation scheme (Carroll, *et al.*, 1998; Lin, 1998) with standard measures: precision, recall and F_1. Relations are organized into a hierarchy with the root node specifying an unlabeled dependency. The microaveraged precision, recall and F_1 scores are calculated from the counts for all relations in the hierarchy which subsume the parser output. The microaveraged F_1 score for the baseline system using this evaluation scheme is 75.61%, which – over similar sets of relational dependencies – is broadly comparable to recent evaluation results published by King and collaborators with their state-of-the-art parsing system (Briscoe *et al.*, 2006).

[3]The pipeline is the same as that used for creating S though we do not automatically map the bracketing to be more consistent with the system grammar, instead, we simply removed unary brackets.

4.1 Wilcoxon Signed Ranks Test

The Wilcoxon Signed Ranks (Wilcoxon, henceforth) test is a *non-parametric* test for statistical significance that is appropriate when there is one data sample and several measures. For example, to compare the accuracy of two parsers over the same data set. As the number of samples (sentences) is large we use the normal approximation for z. Siegel and Castellan (1988) describe and motivate this test. We use a 0.05 level of significance, and provide z-value probabilities for significant results reported below. These results are computed over microaveraged F_1 scores for each sentence in DepBank.

5 Training from Unlabeled Bracketings

We parsed all the bracketed training data using the baseline model to obtain up to 1K top-ranked derivations and found that a significant proportion of the sentences of the potential set available for training had only a single derivation compatible with their unlabeled bracketing. We refer to these sets as the *unambiguous training data* (γ) and will refer to the remaining sentences (for which more than one derivation was compatible with their unlabeled bracketing) as the *ambiguous training data* (α). The availability of significant quantities of unambiguous training data that can be found automatically suggests that we may be able to dispense with the costly reannotation step required to generate the fully supervised training corpus, B.

Table 1 illustrates the split of the corpora into mutually exclusive sets γ, α, 'no match' and 'timeout'. The latter two sets are not utilized during training and refer to sentences for which all parses were inconsistent with the bracketing and those for which no parses were found due to time and memory limitations (self-imposed) on the system.[4] As our grammar is different from that implicit in the WSJ PTB there is a high proportion of sentences where no parses were consistent with the unmodified PTB bracketing. However, a preliminary investigation of no matches didn't yield any clear patterns of inconsistency that we could quickly ameliorate by simple modifications of the PTB bracketing. We leave for the future a more extensive investigation of these cases which, in principle, would allow us to make more use of this training data. An alternative approach that we have also explored is to utilize a similar bootstrapping approach with data partially-annotated for grammatical relations (Watson and Briscoe, 2007).

Corpus	$\mid\gamma\mid$	$\mid\alpha\mid$	No Match	Timeout
S	1097	4138	1322	191
W	6334	15152	15749	1094
SW	7409	19248	16946	1475

Table 1: Corpus split for S, W and SW.

5.1 Confidence-Based Approaches

We use γ to build an initial model. We then utilize this initial model to derive derivations (compatible with the unlabeled partial bracketing) for α from which we select additional training data. We employ two types of selection methods. First, we select the top-ranked derivation only and weight actions which resulted in this derivation equally with those of the initial model (C_1). This method is similar to 'Viterbi training' of HMMs though we do not weight the corresponding actions using the top parse's probability. Secondly, we select more than one derivation, placing an appropriate weight on the corresponding action histories based on the initial model's confidence in the derivation. We consider three such models, in which we weight transitions corresponding to each derivation ranked r with probability p in the set of size n either using $\frac{1}{n}$, $\frac{1}{r}$ or p itself to weight counts.[5] For example, given a treebank T with sentences $t = (s, U)$, function P to return the set of parses consistent with U given t and function A that returns the set of actions given a parse p, then the frequency count of action a_k in C_r is

[4]As there are time and memory restrictions during parsing, the SW results are not equal to the sum of those from S and W analysis.

[5]In §2.1 we calculate action probabilities based on frequency counts where we perform a weighted sum over action histories and each history has a weight of 1. We extend this scheme to include weights that differ between action histories corresponding to each derivation.

determined as follows:

$$| a_k | = \sum_{i=1}^{|T|} \sum_{j=1, a_k \in A(p_{ij})}^{|P(t_i)|} \frac{1}{j}$$

These methods all perform normalization over the resulting action histories using the training function I_L and will be referred to as C_n, C_r and C_p, respectively. C_n is a 'uniform' model which weights counts only by degree of ambiguity and makes no use of ranking information. C_r weights counts by derivation rank, and C_p is simpler than and different to one iteration of EM as outside probabilities are not utilized. All of the semi-supervised functions described here take two arguments: an initial model and the data to train over, respectively.

Models derived from unambiguous training data, γ, alone are relatively accurate, achieving indistinguishable performance to that of the baseline system given either W or SW as training data. We utilize these models as initial models and train over different corpora with each of the confidence-based models. Table 2 gives results for all models. Results statistically significant compared to the baseline system are shown in bold print (better) or italic print (worse). These methods show promise, often yielding systems whose performance is significantly better than the baseline system. Method C_r achieved the best performance in this experiment and remained consistently better in those reported below. Throughout the different approaches a domain effect can be seen, models utilizing just S are worse, although the best performing models benefit from the use of both S and W as training data (i.e. SW).

5.2 EM

Our EM model differs from that of Pereira and Schabes as a PGLR parser adds *context* over a PCFG so that a single rule can be applied in several different states containing reduce actions. Therefore, the summation and normalization performed for a CFG rule within IOA is instead applied within such contexts. We can apply I (our PGLR normalization function without Laplace smoothing) to perform the required steps if we output the action history with the

Model	Prec	Rec	F$_1$	$P(z)^{\ddagger}$
Baseline	77.05	74.22	75.61	-
$I_L(\gamma(S))$	76.02	73.40	*74.69*	0.0294
$C_1(I_L(\gamma(S)), \alpha(S))$	77.05	74.22	75.61	0.4960
$C_n(I_L(\gamma(S)), \alpha(S))$	77.51	74.80	76.13	0.0655
$C_r(I_L(\gamma(S)), \alpha(S))$	77.73	74.98	**76.33**	0.0154
$C_p(I_L(\gamma(S)), \alpha(S))$	76.45	73.91	75.16	0.2090
$I_L(\gamma(W))$	77.01	74.31	75.64	0.1038
$C_1(I_L(\gamma(W)), \alpha(W))$	76.90	74.23	75.55	0.2546
$C_n(I_L(\gamma(W)), \alpha(W))$	77.85	75.07	**76.43**	0.0017
$C_r(I_L(\gamma(W)), \alpha(W))$	77.88	75.04	**76.43**	0.0011
$C_p(I_L(\gamma(W)), \alpha(W))$	77.40	74.75	76.05	0.1335
$I_L(\gamma(SW))$	77.09	74.35	75.70	0.1003
$C_1(I_L(\gamma(SW)), \alpha(SW))$	76.86	74.21	75.51	0.2483
$C_n(I_L(\gamma(SW)), \alpha(SW))$	77.88	75.05	**76.44**	0.0048
$C_r(I_L(\gamma(SW)), \alpha(SW))$	78.01	75.13	**76.54**	0.0007
$C_p(I_L(\gamma(SW)), \alpha(SW))$	77.54	74.95	76.23	0.0618

Table 2: Performance of all models on DepBank. ‡represents the statistical significance of the system against the baseline model.

corresponding normalized inside-outside weight for each node (Watson *et al.*, 2005).

We perform EM starting from two initial models; either a uniform probability model, $I_L()$, or from models derived from unambiguous training data, γ. Figure 1 shows the cross entropy decreasing monotonically from iteration 2 (as guaranteed by the EM method) for different corpora and initial models. Some models show an initial increase in cross-entropy from iteration 1 to iteration 2, because the models are initialized from a subset of the data which is used to perform maximisation. Cross-entropy increases, by definition, as we incorporate ambiguous data with more than one consistent derivation.

Performance over DepBank can be seen in Figures 2, 3, and 4 for each dataset S, W and SW, respectively. Comparing the C_r and EM lines in each of Figures 2, 3, and 4, it is evident that C_r outperforms EM across all datasets, regardless of the initial model applied. In most cases, these results are significant, even when we manually select the best model (iteration) for EM.

The graphs of EM performance from iteration 1 illustrate the same 'classical' and 'initial' patterns observed by Elworthy (1994). When EM is initialized from a relatively poor model, such as that built from S (Figure 2), a 'classical'

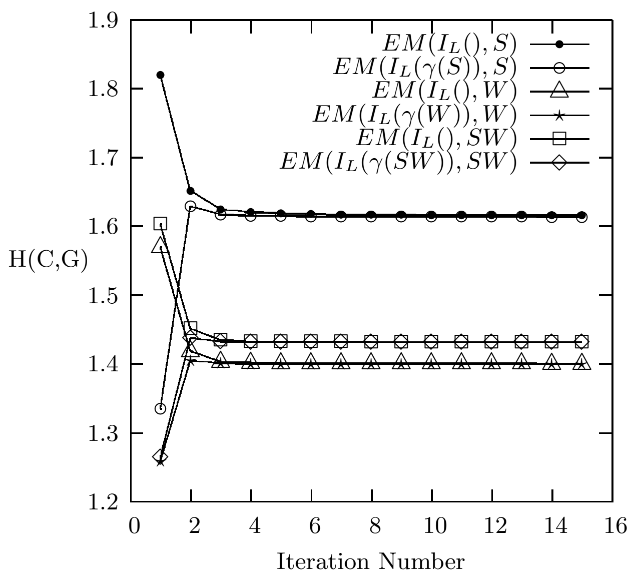


Figure 1: Cross Entropy Convergence for various training data and models, with EM.

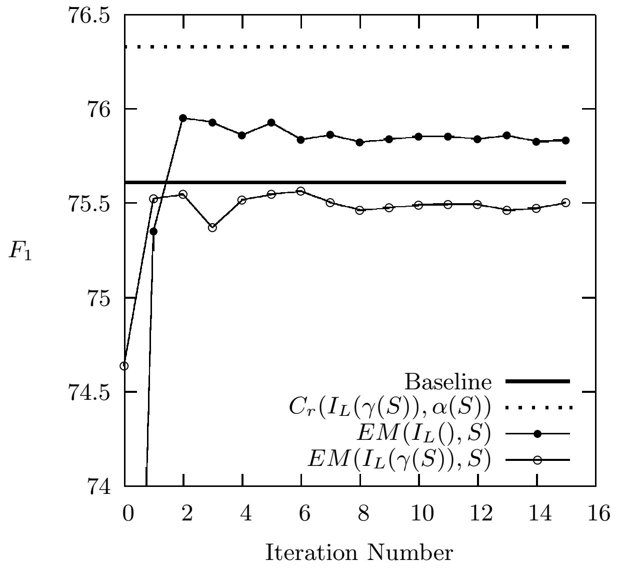


Figure 2: Performance over S for C_r and EM.

pattern emerges with relatively steady improvement from iteration 1 until performance asymptotes. However, when the starting point is better (Figures 3 and 4), the 'initial' pattern emerges in which the best performance is reached after a single iteration.

6 Tuning to a New Domain

When building NLP applications we would want to be able to tune a parser to a new domain with minimal manual effort. To obtain training data in a new domain, annotating a corpus with partial-bracketing information is much cheaper than full annotation. To investigate whether such data would be of value, we considered W to be the corpus over which we were tuning and applied the best performing model trained over S, $C_r(I_L(\gamma(S)), \alpha(S))$, as our initial model. Figure 5 illustrates the performance of C_r compared to EM.

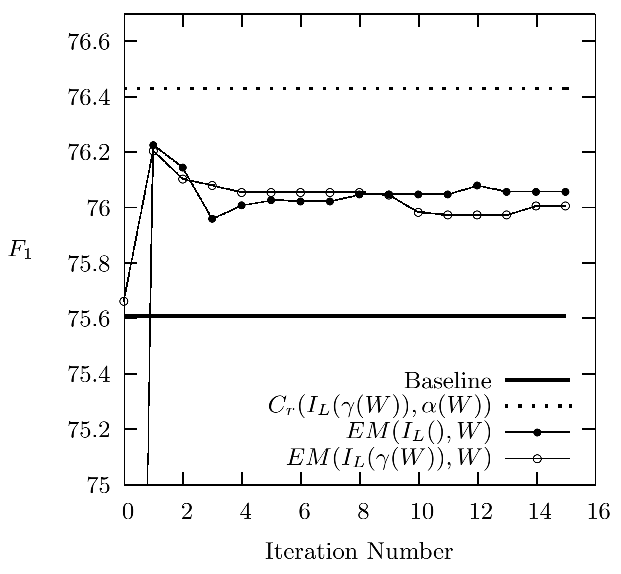


Figure 3: Performance over W for C_r and EM.

Tuning using C_r was not significantly different from the model built directly from the entire data set with C_r, achieving 76.57% as opposed to 76.54% F_1 (see Table 2). By contrast, EM performs better given all the data from the beginning rather than tuning to the new domain.

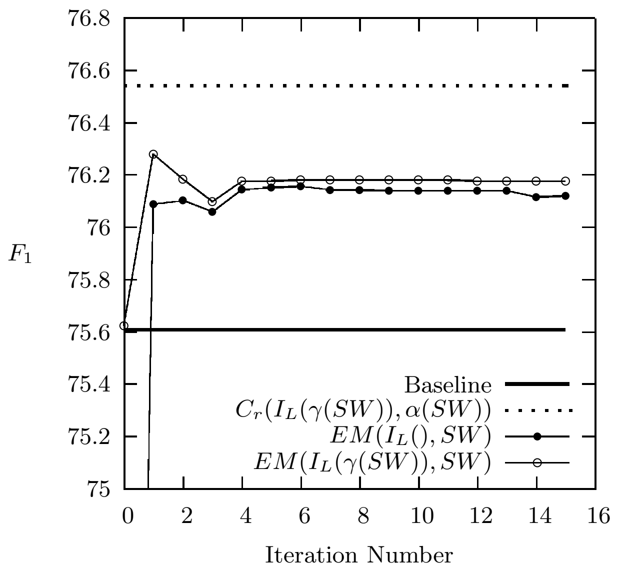


Figure 4: Performance over SW for C_r and EM.

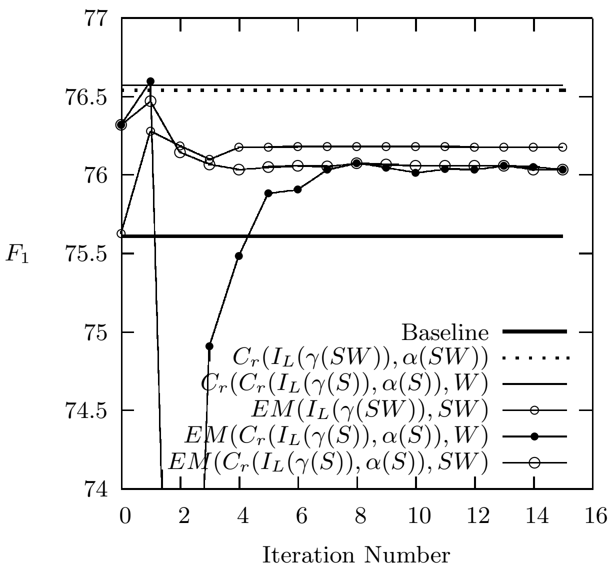


Figure 5: Tuning over the WSJ PTB (W) from Susanne Corpus (S).

C_r generally outperforms EM, though it is worth noting the behavior of EM given only the tuning data (W) rather than the data from both domains (SW). In this case, the graph illustrates a combination of Elworthy's 'initial' and 'classical' patterns. The steep drop in performance (down to 69.93% F_1) after the first iteration is probably due to loss of information from S. However, this run also eventually converges to similar performance, suggesting that the information in S is effectively disregarded as it forms only a small portion of SW, and that these runs effectively converge to a local maximum over W.

Bacchiani *et al.* (2006), working in a similar framework, explore weighting the contribution (frequency counts) of the in-domain and out-of-domain training datasets and demonstrate that this can have beneficial effects. Furthermore, they also tried unsupervised tuning to the in-domain corpus by weighting parses for it by their normalized probability. This method is similar to our C_p method. However, when we tried unsupervised tuning using the WSJ and an initial model built from S in conjunction with our confidence-based methods, performance degraded significantly.

7 Conclusions

We have presented several semi-supervised *confidence-based* training methods which have significantly improved performance over an extant (more supervised) method, while also reducing the manual effort required to create training or tuning data. We have shown that given a medium-sized unlabeled partially bracketed corpus, the confidence-based models achieve superior results to those achieved with EM applied to the same PGLR parse selection model. Indeed, a bracketed corpus provides flexibility as existing treebanks can be utilized despite the incompatibility between the system grammar and the underlying grammar of the treebank. Mapping an incompatible annotated treebank to a compatible partially-bracketed corpus is relatively easy compared to mapping to a compatible fully-annotated corpus.

An immediate benefit of this work is that (re)training parsers with incrementally-modified grammars based on different linguistic frameworks should be much more straightforward – see, for example Oepen *et al.* (2002) for a good discussion of the problem. Furthermore, it suggests that it may be possible to usefully tune a parser to a new domain with less annotation effort.

Our findings support those of Elworthy (1994) and Merialdo (1994) for POS tagging and suggest that EM is not always the most suitable semi-supervised training method (especially when some in-domain training data is available). The confidence-based methods were successful because the level of noise introduced did not outweigh the benefit of incorporating all derivations compatible with the bracketing in which the derivations contained a high proportion of correct constituents. These findings may not hold if the level of bracketing available does not adequately constrain the parses considered – see Hwa (1999) for a related investigation with EM.

In future work we intend to further investigate the problem of tuning to a new domain, given that minimal manual effort is a major priority. We hope to develop methods which required no manual annotation, for example, high precision automatic partial bracketing using phrase chunking and/or named entity recognition techniques might yield enough information to support the training methods developed here.

Finally, further experiments on weighting the contribution of each dataset might be beneficial. For instance, Bacchiani *et al.* (2006) demonstrate imrpovements in parsing accuracy with unsupervised adaptation from unannotated data and explore the effect of different weighting of counts derived from the supervised and unsupervised data.

Acknowledgements

The first author is funded by the Overseas Research Students Awards Scheme, and the Poynton Scholarship awarded by the Cambridge Australia Trust in collaboration with the Cambridge Commonwealth Trust. Development of the RASP system was and is supported by the EPSRC (grants GR/N36462, GR/N36493 and GR/T19919).

References

Bacchiani, M., Riley, M., Roark, B. and R. Sproat (2006) ‘MAP adaptation of stochastic grammars’, *Computer Speech and Language, vol.20.1,* pp.41–68.

Baker, J. K. (1979) ‘Trainable grammars for speech recognition’ in Klatt, D. and Wolf, J. (eds.), *Speech Communications Papers for the 97th Meeting of the Acoustical Society of America,* MIT, Cambridge, Massachusetts, pp. 557–550.

Briscoe, E.J., J. Carroll and R. Watson (2006) ‘The Second Release of the RASP System’, *Proceedings of ACL-Coling’06,* Sydney, Australia.

Carroll, J., Briscoe, T. and Sanfilippo, A. (1998) ‘Parser evaluation: a survey and a new proposal’, *Proceedings of LREC,* Granada, pp. 447–454.

Clark, S. and J. Curran (2004) ‘Parsing the WSJ Using CCG and Log-Linear Models’, *Proceedings of 42nd Meeting of the Association for Computational Linguistics,* Barcelona, pp. 103–110.

Collins, M. (1999) *Head-driven Statistical Models for Natural Language Parsing,* PhD Dissertation, University of Pennsylvania.

Elworthy, D. (1994) ‘Does Baum-Welch Reestimation Help Taggers?’, *Proceedings of ANLP,* Stuttgart, Germany, pp. 53–58.

Gildea, D. (2001) ‘Corpus variation and parser performance’, *Proceedings of EMNLP,* Pittsburgh, PA.

Hockenmaier, J. (2003) *Data and models for statistical parsing with Combinatory Categorial Grammar,* PhD Dissertation, The University of Edinburgh.

Hwa, R. (1999) ‘Supervised grammar induction using training data with limited constituent information’, *Proceedings of ACL,* College Park, Maryland, pp. 73–79.

Inui, K., V. Sornlertlamvanich, H. Tanaka and T. Tokunaga (1997) ‘A new formalization of probabilistic GLR parsing’, *Proceedings* *of IWPT,* MIT, Cambridge, Massachusetts, pp. 123–134.

King, T.H., R. Crouch, S. Riezler, M. Dalrymple and R. Kaplan (2003) 'The PARC700 Dependency Bank', *Proceedings of LINC,* Budapest.

Lin, D. (1998) 'Dependency-based evaluation of MINIPAR', *Proceedings of Workshop at LREC'98 on The Evaluation of Parsing Systems,* Granada, Spain.

McClosky, D., Charniak, E. and M. Johnson (2006) 'Effective self-training for parsing', *Proceedings of HLT-NAACL,* New York.

Merialdo, B. (1994) 'Tagging English Text with a Probabilistic Model', *Computational Linguistics, vol.20.2,* pp.155–171.

Miyao, Y. and J. Tsujii (2002) 'Maximum Entropy Estimation for Feature Forests', *Proceedings of HLT,* San Diego, California.

Oepen, S., K. Toutanova, S. Shieber, C. Manning, D. Flickinger, and T. Brants (2002) 'The LinGO Redwoods Treebank: Motivation and preliminary applications', *Proceedings of COLING,* Taipei, Taiwan.

Pereira, F and Y. Schabes (1992) 'Inside-Outside Reestimation From Partially Bracketed Corpora', *Proceedings of ACL,* Delaware.

Prescher, D. (2001) 'Inside-outside estimation meets dynamic EM', *Proceedings of 7th Int. Workshop on Parsing Technologies (IWPT01),* Beijing, China.

Riezler, S., T. King, R. Kaplan, R. Crouch, J. Maxwell III and M. Johnson (2002) 'Parsing the Wall Street Journal using a Lexical-Functional Grammar and Discriminative Estimation Techniques', *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics,* Philadelphia, pp. 271–278.

Sampson, G. (1995) *English for the Computer,* Oxford University Press, Oxford, UK.

Siegel S. and N. J. Castellan (1988) *Nonparametric Statistics for the Behavioural Sciences, 2nd edition,* McGraw-Hill.

Tadayoshi, H., Y. Miyao and J. Tsujii (2005) 'Adapting a probabilistic disambiguation model of an HPSG parser to a new domain', *Proceedings of IJCNLP,* Jeju Island, Korea.

Tomita, M. (1987) 'An Efficient Augmented Context-Free Parsing Algorithm', *Computational Linguistics, vol.13(1–2),* pp.31–46.

Watson, R. and E.J. Briscoe (2007) 'Adapting the RASP system for the CoNLL07 domain-adaptation task', *Proceedings of EMNLP-CoNLL-07,* Prague.

Watson, R., J. Carroll and E.J. Briscoe (2005) 'Efficient extraction of grammatical relations', *Proceedings of 9th Int. Workshop on Parsing Technologies (IWPT'05),* Vancouver, Ca..

Adapting WSJ-Trained Parsers to the British National Corpus Using In-Domain Self-Training

Jennifer Foster, Joachim Wagner, Djamé Seddah and Josef van Genabith
National Centre for Language Technology
School of Computing, Dublin City University, Dublin 9, Ireland
{jfoster, jwagner, josef}@computing.dcu.ie, dseddah@paris4.sorbonne.fr*

Abstract

We introduce a set of 1,000 gold standard parse trees for the British National Corpus (BNC) and perform a series of self-training experiments with Charniak and Johnson's reranking parser and BNC sentences. We show that retraining this parser with a combination of one million BNC parse trees (produced by the same parser) and the original WSJ training data yields improvements of 0.4% on WSJ Section 23 and 1.7% on the new BNC gold standard set.

1 Introduction

Given the success of statistical parsing models on the Wall Street Journal (WSJ) section of the Penn Treebank (PTB) (Charniak, 2000; Collins, 2003, for example), there has been a change in focus in recent years towards the problem of replicating this success on genres other than American financial news stories. The main challenge in solving the parser adaptation problem are the resources required to construct reliable annotated training examples.

A breakthrough has come in the form of research by McClosky et al. (2006a; 2006b) who show that self-training can be used to improve parser performance when combined with a two-stage reranking parser model (Charniak and Johnson, 2005). Self-training is the process of training a parser on its own output, and earlier self-training experiments using generative statistical parsers did not yield encouraging results (Steedman et al., 2003). McClosky et al. (2006a; 2006b) proceed as follows: sentences from the *LA Times* newspaper are parsed by a first-stage generative statistical parser trained on some seed training data (WSJ Sections 2-21) and the *n*-best parse trees produced by this parser are reranked by a discriminative reranker. The highest ranked parse trees are added to the training set of the parser and the parser is retrained. This self-training method gives improved performance, not only on Section 23 of the WSJ (an absolute f-score improvement of 0.8%), but also on test sentences from the Brown corpus (Francis and Kučera, 1979) (an absolute f-score improvement of 2.6%).

In the experiments of McClosky et al. (2006a; 2006b), the parse trees used for self-training come from the same domain (American newspaper text) as the parser's original seed training material. Bacchiani et al. (2006) find that self-training is effective when the parse trees used for self-training (WSJ parse trees) come from a different domain to the seed training data and from the same domain as the test data (WSJ sentences). They report a performance boost of 4.2% on WSJ Section 23 for a generative statistical parser trained on Brown seed data when it is self-trained using 200,000 WSJ parse trees. However, McCloskey et al. (2006b) report a drop in performance for their reranking parser when the experiment is repeated in the opposite direction, i.e. with Brown data for self-training and testing, and WSJ data for seed training. In contrast, we report successful in-domain[1] self-training experiments with the BNC data as self-training and test material, *and* with the WSJ-trained reranking parser used by McCloskey et al. (2006a; 2006b).

We parse the BNC (Burnard, 2000) in its entirety

*Now affiliated to Lalic, Université Paris 4 La Sorbonne.

[1]We refer to data as being *in-domain* if it comes from the same domain as the test data and *out-of-domain* if it does not.

using the reranking parser of Charniak and Johnson (2005). 1,000 BNC sentences are manually annotated for constituent structure, resulting in the first gold standard set for this corpus. The gold standard set is split into a development set of 500 parse trees and a test set of 500 parse trees and used in a series of self-training experiments: Charniak and Johnson's parser is retrained on combinations of WSJ treebank data and its own parses of BNC sentences. These combinations are tested on the BNC development set and Section 00 of the WSJ. An optimal combination is chosen which achieves a Parseval labelled bracketing f-score of 91.7% on Section 23 and 85.6% on the BNC gold standard test set. For Section 23 this is an absolute improvement of 0.4% on the baseline results of this parser, and for the BNC data this is a statistically significant improvement of 1.7%.

2 The BNC Data

The BNC is a 100-million-word balanced part-of-speech-tagged corpus of written and transcribed spoken English. Written text comprises 90% of the BNC: 75% non-fictional and 25% fictional. To facilitate parsing with a WSJ-trained parser, some reversible transformations were applied to the BNC data, e.g. British English spellings were converted to American English and neutral quotes disambiguated. The reranking parser of Charniak and Johnson (2005) was used to parse the BNC. 99.8% of the 6 million BNC sentences obtained a parse, with an average parsing speed of 1.4s per sentence.

A gold standard set of 1,000 BNC sentences was constructed by one annotator by correcting the output of the first stage of Charniak and Johnson's reranking parser. The sentences included in the gold standard were chosen at random from the BNC, subject to the condition that they contain a verb which does not occur in the training sections of the WSJ section of the PTB (Marcus et al., 1993). A decision was made to select sentences for the gold standard set which differ from the sentences in the WSJ training sections, and one way of finding different sentences is to focus on verbs which are not attested in the WSJ Sections 2-21. It is expected that these gold standard parse trees can be used as training data although they are used only as test and development data in this work. Because they contain verbs which do not occur in the parser's training set, they are likely to represent a hard test for WSJ-trained parsers. The PTB bracketing guidelines (Bies et al., 1995) and the PTB itself were used as references by the BNC annotator. Functional tags and traces were not annotated. The annotator noticed that the PTB parse trees sometimes violate the PTB bracketing guidelines, and in these cases, the annotator chose the analysis set out in the guidelines. It took approximately 60 hours to build the gold standard set.

3 Self-Training Experiments

Charniak and Johnson's reranking parser (June 2006 version) is evaluated against the BNC gold standard development set. Labelled precision (LP), recall (LR) and f-score measures[2] for this parser are shown in the first row of Table 1. The f-score of 83.7% is lower than the f-score of 85.2% reported by McClosky et al. (2006b) for the same parser on Brown corpus data. This difference is reasonable since there is greater domain variation between the WSJ and the BNC than between the WSJ and the Brown corpus, and all BNC gold standard sentences contain verbs not attested in WSJ Sections 2-21.

We retrain the first-stage generative statistical parser of Charniak and Johnson using combinations of BNC trees (parsed using the reranking parser) and WSJ treebank trees. We test the combinations on the BNC gold standard development set and on WSJ Section 00. Table 1 shows that parser accuracy increases with the size of the in-domain self-training material.[3] The figures confirm the claim of McClosky et al. (2006a) that *self-training* with a reranking parsing model is effective for improving parser accuracy in general, and the claim of Gildea (2001) that *training* on in-domain data is effective for parser adaption. They confirm that *self-training* on *in-domain* data is effective for parser adaptation. The WSJ Section 00 results suggest that, in order to maintain performance on the seed training domain, it is necessary to combine BNC parse trees

[2]All scores are for the second stage of the parsing process, i.e. the evaluation takes place after the reranking. All evaluation is carried out using the Parseval labelled bracketing metrics, with `evalb` and parameter file `new.prm`.

[3]The notation *bnc500K+5wsj* refers to a set of 500,000 parser output parse trees of sentences taken randomly from the BNC concatenated with five copies of WSJ Sections 2-21.

	BNC Development			WSJ Section 00		
Self-Training	LP	LR	LF	LP	LR	LF
-	83.6	83.7	83.7	91.6	90.5	91.0
bnc50k	83.7	83.7	83.7	90.0	88.0	89.0
bnc50k+1wsj	84.4	84.4	84.4	91.6	90.3	91.0
bnc250k	84.7	84.5	84.6	91.1	89.3	90.2
bnc250k+5wsj	85.0	84.9	85.0	91.8	90.5	91.2
bnc500k+5wsj	85.2	85.1	85.2	91.9	90.4	91.2
bnc500k+10wsj	85.1	85.1	85.1	91.9	90.6	91.2
bnc1000k+5wsj	86.5	86.2	86.3	91.7	90.3	91.0
bnc1000k+10wsj	86.1	85.9	**86.0**	92.0	90.5	**91.3**
bnc1000k+40wsj	85.5	85.5	85.5	91.9	90.6	91.3
	BNC Test			**WSJ Section 23**		
-	84.0	83.7	83.9	91.8	90.9	91.3
bnc1000k+10wsj	85.7	85.4	**85.6**	92.3	91.1	**91.7**

Table 1: In-domain Self-Training Results

with the original seed training material during the self-training phase.

Of the self-training combinations with above-baseline improvements for both development sets, the combination of 1,000K BNC parse trees and Section 2-21 of the WSJ (multiplied by ten) yields the highest improvement for the BNC data, and we present final results with this combination for the BNC gold standard test set and WSJ Section 23. There is an absolute improvement on the original reranking parser of 1.7% on the BNC gold standard test set and 0.4% on WSJ Section 23. The improvement on BNC data is statistically significant for both precision and recall (p $<$ 0.0002, p $<$ 0.0002). The improvement on WSJ Section 23 is statistically significant for precision only (p $<$ 0.003).

4 Conclusion and Future Work

We have introduced a set of 1,000 gold standard parse trees for the BNC. We have performed self-training experiments with Charniak and Johnson's reranking parser and sentences from the BNC. We have shown that retraining this parser with a combination of one million BNC parse trees (produced by the same parser) and the original WSJ training data yields improvements of 0.4% on WSJ Section 23 and 1.7% on the BNC gold standard sentences. These results indicate that self-training on in-domain data can be used for parser adaptation.

Our BNC gold standard set consists of sentences containing verbs which are not in the WSJ training sections. We suspect that this makes the gold standard set a hard test for WSJ-trained parsers, and our results are likely to represent a lower bound for WSJ-trained parsers on BNC data. When used as training data, we predict that the novel verbs in the BNC gold standard set add to the variety of training material, and will further help parser adaptation from the WSJ domain – a matter for further research.

Acknowledgments We thank the IRCSET Embark Initiative (basic research grant SC/02/298 and postdoctoral fellowship P/04/232), Science Foundation Ireland (Principal Investigator grant 04/IN.3/I527) and the Irish Centre for High End Computing for supporting this research.

References

Michiel Bacchiani, Michael Riley, Brian Roark, and Richard Sproat. 2006. Map adaptation of stochastic grammars. *Computer Speech and Language*, 20(1):41–68.

Ann Bies, Mark Ferguson, Karen Katz, and Robert MacIntyre. 1995. Bracketing guidelines for treebank II style, Penn Treebank project. Technical Report MS-CIS-95-06, University of Pennsylvania.

Lou Burnard. 2000. User reference guide for the British National Corpus. Technical report, Oxford University.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best-parsing and maxent discriminative reranking. In *Proceedings of ACL-05*, pages 173–180, Barcelona.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of NAACL-00*, pages 132–139, Seattle.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):499–637.

W. Nelson Francis and Henry Kučera. 1979. Brown Corpus Manual. Technical report, Brown University, Providence.

Daniel Gildea. 2001. Corpus variation and parser performance. In *Proceedings of EMNLP-01*, pages 167–202, Barcelona.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330.

David McClosky, Eugene Charniak, and Mark Johnson. 2006a. Effective self-training for parsing. In *Proceedings of HLT-NAACL-06*, pages 152–159, New York.

David McClosky, Eugene Charniak, and Mark Johnson. 2006b. Reranking and self-training for parser adaptation. In *Proceedings of COLING-ACL-06*, pages 337–344, Sydney.

Mark Steedman, Miles Osbourne, Anoop Sarkar, Stephen Clark, Rebecca Hwa, Julia Hockenmaier, Paul Ruhlen, Steven Baker, and Jeremiah Crim. 2003. Boot-strapping statistical parsers from small datasets. In *Proceedings of EACL-03*, Budapest.

The Impact of Deep Linguistic Processing on Parsing Technology

Timothy Baldwin
University of Melbourne
tim@csse.unimelb.edu.au

Mark Dras
Macquarie University
madras@ics.mq.edu.au

Julia Hockenmaier
University of Pennsylvania
juliahr@cis.upenn.edu

Tracy Holloway King
PARC
thking@parc.com

Gertjan van Noord
University of Groningen
vannoord@let.rug.nl

Abstract

As the organizers of the ACL 2007 Deep Linguistic Processing workshop (Baldwin et al., 2007), we were asked to discuss our perspectives on the role of current trends in deep linguistic processing for parsing technology. We are particularly interested in the ways in which efficient, broad coverage parsing systems for linguistically expressive grammars can be built and integrated into applications which require richer syntactic structures than shallow approaches can provide. This often requires hybrid technologies which use shallow or statistical methods for pre- or post-processing, to extend coverage, or to disambiguate the output.

1 Introduction

Our talk will provide a view on the relevance of deep linguistic processing for parsing technologies from the perspective of the organizers of the ACL 2007 Workshop on Deep Linguistic Processing (Baldwin et al., 2007). The workshop was conceived with the broader aim of bringing together the different computational linguistic sub-communities which model language predominantly by way of theoretical syntax, either in the form of a particular theory (e.g. CCG, HPSG, LFG, TAG, the Prague School) or a more general framework which draws on theoretical and descriptive linguistics. These "deep linguistic processing" approaches differ from shallower methods in that they yield richer, more expressive, structural representations which capture long-distance dependencies or the underlying predicate-argument structure directly.

Aspects of this research have often had their own separate fora, such as the ACL 2005 workshop on deep lexical acquisition (Baldwin et al., 2005), as well as the TAG+ (Kallmeyer and Becker, 2006), Alpino (van der Beek et al., 2005), ParGram (Butt et al., 2002) and DELPH-IN (Oepen et al., 2002) projects and meetings. However, the fundamental approaches to building a linguistically-founded system and many of the techniques used to engineer efficient systems are common across these projects and independent of the specific grammar formalism chosen. As such, we felt the need for a common meeting in which experiences could be shared among a wider community, similar to the role played by recent meetings on grammar engineering (Wintner, 2006; Bender and King, 2007).

2 The promise of deep parsing

Deep linguistic processing has traditionally been concerned with grammar development (for use in both parsing and generation). However, the linguistic precision and complexity of the grammars meant that they had to be manually developed and maintained, and were computationally expensive to run.

In recent years, machine learning approaches have fundamentally altered the field of natural language processing. The availability of large, manually annotated, treebanks (which typically take years of prior linguistic groundwork to produce) enabled the rapid creation of robust, wide-coverage parsers. However, the standard evaluation metrics for which such parsers have been optimized generally ignore much of the rich linguistic information in the original treebanks. It is therefore perhaps only natural that deep processing methods, which often require substantial amounts of manual labor, have received considerably less attention during this period.

But even if further work is required for deep processing techniques to fully mature, we believe that applications that require natural language understanding or inference, among others, will ultimately need detailed syntactic representations (capturing, e.g., bounded and unbounded long-range dependencies) from which semantic interpretations can easily be built. There is already some evidence that our current deep techniques can, in some cases, outperform shallow approaches. There has been work demonstrating this in question answering, targeted information extraction and the recent textual entailment recognition task, and perhaps most notably in machine translation: in this latter field, after a period of little use of linguistic knowledge, deeper techniques are beginning to lead to better performance, e.g. by redefining phrases by syntactic "treelets" rather than contiguous word sequences, or by explicitly including a syntactic component in the probability model, or by syntactic preprocessing of the data.

3 Closing the divide

In the past few years, the divide between "deep", rule-based, methods and "shallow", statistical, approaches, has begun to close from both sides. Recent advances in using the same treebanks that have advanced shallow techniques to extract more expressive grammars or to train statistical disambiguators for them, and in developing framework-specific treebanks, have made it possible to obtain similar coverage, robustness, and disambiguation accuracy for parsers that use richer structural representations. As witnessed by many of the papers in our workshop (Baldwin et al., 2007), a large proportion of current deep systems have statistical components to them, e.g., as pre- or post-processing to control ambiguity, as means of acquiring and extending lexical resources, or even use machine learning techniques to acquire deep grammars automatically. From the other side of the divide, many of the purely statistical approaches are using progressively richer linguistic features and are taking advantage of these more expressive features to tackle problems that were traditionally thought to require deep systems, such as the recovery of traces or semantic roles.

4 The continued need for research on deep processing

Although statistical techniques are becoming commonplace even for systems built around handwritten grammars, there is still a need for further linguistic research and manual grammar development. For example, supervised machine-learning approaches rely on large amounts of manually annotated data. Where such data are available, developers of deep parsers and grammars can exploit them to determine frequency of certain constructions, to bootstrap gold standards for their systems, and to provide training data for the statistical components of their systems such as parse disambiguators. But for the majority of the world's languages, and even for many languages with large numbers of speakers, such corpora are unavailable. Under these circumstances, manual grammar development is unavoidable, and recent progress has allowed the underlying systems to become increasingly better engineered, allowing for more rapid development of any given grammar, as well as for overlay grammars that adapt to particular domains and applications and for porting of grammars from one language to another.

Despite recent work on (mostly dependency grammar-based) multilingual parsing, it is still the case that most research on statistical parsing is done on English, a fixed word-order language where simple context-free approximations are often sufficient. It is unclear whether our current models and algorithms carry over to morphologically richer languages with more flexible word order, and it is possible that the more complex structural representations allowed by expressive formalisms will cease to remain a luxury.

Further research is required on all aspects of deep linguistic processing, including novel linguistic analyses and implementations for different languages, formal comparisons of different frameworks, efficient parse and learning algorithms, better statistical models, innovative uses of existing data resources, and new evaluation tools and methodologies. We were fortunate to receive so many high-quality submissions on all of these topics for our workshop.

5 Conclusion and outlook

Deep linguistic processing brings together a range of perspectives. It covers current approaches to grammar development and issues of theoretical linguistic and algorithmic properties, as well as the application of deep linguistic techniques to large-scale applications such as question answering and dialog systems. Having industrial-scale, efficient parsers and generators opens up new application domains for natural language processing, as well as interesting new ways in which to approach existing applications, e.g., by combining statistical and deep processing techniques in a triage process to process massive data quickly and accurately at a fine level of detail. Notably, several of the papers addressed the relationship of deep linguistic processing to topical statistical approaches, in particular in the area of parsing. There is an increasing interest in deep linguistic processing, an interest which is buoyed by the realization that new, often hybrid, techniques combined with highly engineered parsers and generators and state-of-the-art machines opens the way towards practical, real-world application of this research. We look forward to further opportunities for the different computational linguistic subcommunities who took part in this workshop, and others, to continue to come together in the future.

References

Timothy Baldwin, Anna Korhonen, and Aline Villavicencio, editors. 2005. *Proceedings of the ACL-SIGLEX Workshop on Deep Lexical Acquisition*. Ann Arbor, USA.

Timothy Baldwin, Mark Dras, Julia Hockenmaier, Tracy Holloway King, and Gertjan van Noord, editors. 2007. *Proceedings of the ACL Workshop on Deep Linguistic Processing*, Prague, Czech Republic.

Emily Bender and Tracy Holloway King, editors. 2007. *Grammar Engineering Across Frameworks*, Stanford University. CSLI On-line Publications. to appear.

Miriam Butt, Helge Dyvik, T. H. King, Hiroshi Masuichi, and Christian Rohrer. 2002. The parallel grammar project. In *COLING Workshop on Grammar Engineering and Evaluation*, Taipei, Taiwan.

Laura Kallmeyer and Tilman Becker, editors. 2006. *Proceedings of the Eighth International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+)*, Sydney, Australia.

Stephan Oepen, Dan Flickinger, J. Tsujii, and Hand Uszkoreit, editors. 2002. *Collaborative Language Engineering: A Case Study in Efficient Grammar-based Processing*. CSLI Publications.

Leonoor van der Beek, Gosse Bouma, Jan Daciuk, Tanja Gaustad, Robert Malouf, Mark-Jan Nederhof, Gertjan van Noord, Robbert Prins, and Bego na Villada Moirón. 2005. Algorithms for linguistic processing. NWO Pionier fi nal report. Technical report, University of Groningen.

Shuly Wintner. 2006. Large-scale grammar development and grammar engineering. Research workshop of the Israel Science Foundation.

Improving the Efficiency of a Wide-Coverage CCG Parser

Bojan Djordjevic and **James R. Curran**
School of Information Technologies
University of Sydney
NSW 2006, Australia
{bojan,james}@it.usyd.edu.au

Stephen Clark
Computing Laboratory
Oxford University
Wolfson Building, Parks Road
Oxford, OX1 3QD, UK
stephen.clark@comlab.ox.ac.uk

Abstract

The C&C CCG parser is a highly efficient linguistically motivated parser. The efficiency is achieved using a tightly-integrated supertagger, which assigns CCG lexical categories to words in a sentence. The integration allows the parser to request more categories if it cannot find a spanning analysis. We present several enhancements to the CKY chart parsing algorithm used by the parser. The first proposal is *chart repair*, which allows the chart to be efficiently updated by adding lexical categories individually, and we evaluate several strategies for adding these categories. The second proposal is to add *constraints* to the chart which require certain spans to be constituents. Finally, we propose *partial* beam search to further reduce the search space. Overall, the parsing speed is improved by over 35% with negligible loss of accuracy or coverage.

1 Introduction

A recent theme in parsing research has been the application of statistical methods to linguistically motivated grammars, for example LFG (Kaplan et al., 2004; Cahill et al., 2004), HPSG (Toutanova et al., 2002; Malouf and van Noord, 2004), TAG (Sarkar and Joshi, 2003) and CCG (Hockenmaier and Steedman, 2002; Clark and Curran, 2004b). The attraction of linguistically motivated parsers is the potential to produce rich output, in particular the predicate-argument structure representing the underlying meaning of a sentence. The disadvantage of such parsers is that they are typically not very efficient, parsing a few sentences per second on commodity hardware (Kaplan et al., 2004). The C&C CCG parser (Clark and Curran, 2004b) is an order of magnitude faster, but is still limited to around 25 sentences per second.

The key to efficient CCG parsing is a finite-state *supertagger* which performs much of the parsing work (Bangalore and Joshi, 1999). CCG is a lexicalised grammar formalism, in which elementary syntactic structures — in CCG's case *lexical categories* expressing subcategorisation information — are assigned to the words in a sentence. CCG supertagging can be performed accurately and efficiently by a Maximum Entropy tagger (Clark and Curran, 2004a). Since the lexical categories contain so much grammatical information, assigning them with low average ambiguity leaves the parser, which combines them together, with much less work to do at parse time. Hence Bangalore and Joshi (1999), in the context of LTAG parsing, refer to supertagging as *almost parsing*.

Clark and Curran (2004a) presents a novel method of integrating the supertagger and parser: initially only a small number of categories, on average, is assigned to each word, and the parser attempts to find a spanning analysis using the CKY chart-parsing algorithm. If one cannot be found, the parser requests more categories from the supertagger and builds the chart again from scratch. This process repeats until the parser is able to build a chart containing a spanning analysis.[1]

[1]Tsuruoka and Tsujii (2004) investigate a similar idea in the context of the CKY algorithm for a PCFG.

The supertagging accuracy is high enough that the parser fails to find a spanning analysis using the initial category assignment in approximately 4% of Wall Street Journal sentences (**?**). However, parsing this 4%, which largely consists of the longer sentences, is disproportionately expensive.

This paper describes several modifications to the C&C parser which improve parsing efficiency without reducing accuracy or coverage by reducing the impact of the longer sentences. The first involves *chart repair*, where the CKY chart is repaired when extra lexical categories are added (according to the scheme described above), instead of being rebuilt from scratch. This allows an even tighter integration of the supertagger, in that the parser is able to request *individual* categories. We explore methods for choosing which individual categories to add, resulting in an 11% speed improvement.

The next modification involves parsing with *constraints*, so that certain spans are required to be constituents. This reduces the search space considerably by eliminating a large number of constituents which cross the boundaries of these spans. The best set of constraints results in a 10% speed improvement over the original parser. These constraints are general enough that they could be applied to any constituency-based parser. Finally, we experiment with several beam strategies to reduce the search space, finding that a *partial beam* which operates on part of the chart is most effective, giving a further 6.1% efficiency improvement.

The chart repair and constraints interact in an interesting, and unexpected, manner when combined, giving a 35.7% speed improvement overall without any loss in accuracy or coverage. This speed improvement is particularly impressive because it involves techniques which only apply to 4% of Wall Street Journal sentences.

2 The CCG Parser

Clark and Curran (2004b) describes the CCG parser. The grammar used by the parser is extracted from CCGbank, a CCG version of the Penn Treebank (Hockenmaier, 2003). The grammar consists of 425 lexical categories plus a small number of combinatory rules which combine the categories (Steedman, 2000). A Maximum Entropy supertagger first assigns lexical categories to the words in a sentence, which are then combined by the parser using the combinatory rules. A log-linear model scores the alternative parses. We use the normal-form model, which assigns probabilities to single derivations based on the normal-form derivations in CCGbank. The features in the model are defined over local parts of the derivation and include word-word dependencies. A packed chart representation allows efficient decoding, with the Viterbi algorithm finding the most probable derivation.

The supertagger uses a log-linear model to define a distribution over the lexical category set for each word and the previous two categories (Ratnaparkhi, 1996) and the forward backward algorithm efficiently sums over all histories to give a distribution for each word. These distributions are then used to assign a set of lexical categories to each word (**?**). The number of categories in each set is determined by a parameter β: all categories are assigned whose forward-backward probabilities are within β of the highest probability category (**?**). If the parser cannot then find a spanning analysis, the value of β is reduced — so that more lexical categories are assigned — and the parser tries again. This process repeats until an analysis spanning the whole sentence is found.

In our previous work, when the parser was unable to find a spanning analysis, the chart was destroyed and then rebuilt from scratch with more lexical categories assigned to each word. However, this rebuilding process is wasteful because the new chart is always a superset of the old one and could be created by just updating the previous chart. We describe the *chart repair* process in Section 3 which allows additional categories to be assigned to an existing chart and the CKY algorithm run over just those parts of the chart which require modification.

2.1 Chart Parsing

The parser uses the CKY chart parsing algorithm (Kasami, 1965; Younger, 1967) described in Steedman (2000). The CKY algorithm applies naturally to CCG since the grammar is binary. It builds the chart bottom-up, starting with the lexical categories spanning single words, incrementally increasing the span until the whole sentence is covered. Since the constituents are built in order of span size, at every stage all the sub-constituents which could be used to create a particular new constituent are already present in the chart.

The charts are *packed* by grouping together equivalent chart entries, which allows a large number of derivations to be represented efficiently. Entries are *equivalent* when they interact in the same manner with both the generation of subsequent parse structure and the statistical parse selection. In practice, this means that equivalent entries have the same span; form the same structures, i.e. the remaining derivation plus dependencies, in any subsequent parsing; and generate the same features in any subsequent parsing.

The Viterbi algorithm is used to find the most probable derivation from a packed chart. For each equivalence class of individual entries, we record the entry at the root of the subderivation which has the highest score for the class. The equivalence classes are defined so that any other individual entry cannot be part of the highest scoring derivation for the sentence. The highest-scoring subderivations can be calculated recursively using the highest-scoring equivalence classes that were combined to create the individual entry.

Given a sentence of n words, we define $pos \in \{0, \ldots, n-1\}$ to be the starting position of an entry in the chart (represented by a CCG category) and $span \in \{1, \ldots, n\}$ its length. Let $\mathrm{cell}(pos, span)$ be the set of categories which span the sentence from pos to $pos + span$. These will be combinations of categories in $\mathrm{cell}(pos, k)$ and $\mathrm{cell}(pos{+}k, span{-}k)$ for all $k \in \{1, \ldots, span-1\}$. The chart is a two dimensional array indexed by pos and $span$. The valid $(pos, span)$ pairs correspond to $pos + span \leq n$, that is, to spans that do not extend beyond the end of the sentence. The squares represent valid cells in Figure 1. The span from position 3 with length 4, i.e. $\mathrm{cell}(3, 4)$, is marked with a diamond in Figure 2.

3 Chart Repair

The parser interacts with the supertagger by decreasing the value of the β parameter when a spanning analysis cannot be found for a sentence. This has the effect of adding more lexical categories to the chart. Instead of rebuilding the chart from scratch when new categories are added, it can be repaired

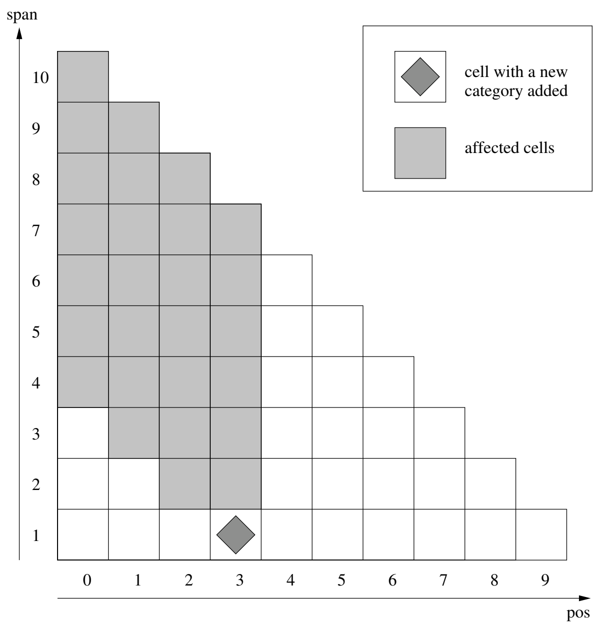


Figure 1: Cells affected by chart repair.

by modifying cells that are affected by the new categories. Considering the case where a single lexical category is added to the ith word in an n word sentence, the new category can only affect the cells that satisfy $pos \leq i$ and $pos + span > i$. These cells are shown in Figure 1 for the word at position 3.

The number of affected cells is $(n{-}pos)(pos{+}1)$, and so the average over the sentence is approximately $\frac{1}{n}\int_0^{n-1}(n-p)(p+1)\ dp \approx \frac{n^2}{6}$ cells. The total number of cells in the chart is $\frac{n(n+1)}{2}$. The chart can therefore be repaired bottom up, in CKY order, by updating a third of the cells on average.

Additional lexical categories for a word are inserted into the corresponding cell in the bottom row, with the additional categories being marked as new. For each cell C in the second row, each pair of cells A and B is considered whose spans combine to create the span of C. In the original CKY, all categories from A are combined with all categories from B. In chart repair, categories are only combined if at least one of them is new, because otherwise the result is already in C. The categories added to C are marked, and the process is repeated for all affected cells in CKY order.

Chart repair speeds up parsing for two reasons. First, it reuses previous computations and eliminates wasteful rebuilding of the chart. Second, it allows lexical categories to be added to the chart *one at a*

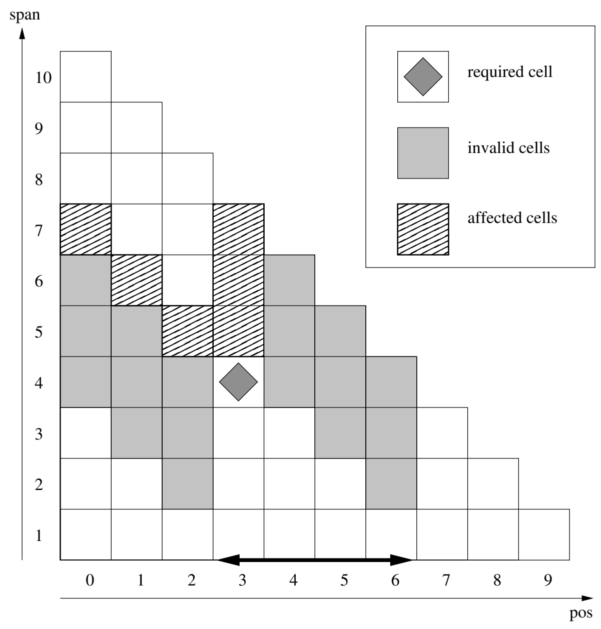


Figure 2: Cells affected by adding a constraint.

time until a spanning derivation is found. In the original approach extra categories were added in bulk by changing the β level, which significantly increased the average ambiguity. Chart repair allows the minimum amount of ambiguity to be added for a spanning derivation to be found.

The C&C parser has a predefined limit on the number of categories in the chart. If this is exceeded before a spanning analysis is found then the parser fails on the sentence. Our new strategy allows a chart containing a spanning analysis to be built with the minimum number of categories possible. This means that some sentences can now be parsed that would have previously exceeded the limit, slightly increasing coverage.

3.1 Category selection

The order in which lexical categories are added to the chart will impact on parsing speed and accuracy, and so we evaluate several alternatives. The first ordering (β VALUE) is by decreasing β value, where the β value is the ratio between the probability of the most likely category and the probability of the given category for that word.[2] The second ordering (PROB) is by decreasing category probability as assigned by the supertagger using the forward-backward algorithm.

We also investigated ordering categories using information from the chart. Examining the sentences which required chart repair showed that, when a word is missing the correct category, the cells affected (as defined in Section 3) by the cell are often empty. The CHART ordering uses this observation to select the next lexical category to assign. It selects the word corresponding to the cell with the highest number of empty affected cells, and then adds the highest probability category not in the chart for that word. Finally, we included a RANDOM ordering baseline for comparison purposes.

4 Constraints

The set of possible derivations can be constrained if we know in advance that a particular span is *required* to be the yield of a single constituent in the correct parse. A *constraint* on span p reduces the search space because p must be the yield of a single cell. This means that cells with yields that cross the boundary of p cannot be part of a correct derivation, and do not need to be considered (the grey cells in Figure 2). In addition, if a cell yields p as a *prefix* or *suffix* (the hashed cells in Figure 2) then it also has constraints on how it can be created.

Figure 2 shows an example constraint requiring words 3–6 to be a constituent, which corresponds to $p = \text{cell}(3, 4)$. Consider $\text{cell}(3, 7)$: it yields words 3–9 and so contains p as the prefix. Normally it can be created by combining $\text{cell}(3, 1)$ with $\text{cell}(4, 6)$, or $\text{cell}(3, 2)$ with $\text{cell}(5, 5)$, and so on up to $\text{cell}(3, 6)$ with $\text{cell}(9, 1)$. However the first three combinations are not allowed because the second child crosses the boundary of p. This gives a lower limit for the span of the left child. Similarly, if p is the suffix of the span of a cell then there is a lower limit on the span of the right child.

As the example demonstrates, a single constraint can eliminate many combinations, reducing the search space significantly, and thus improving parsing efficiency.

4.1 Creating Constraints

How can we know in advance that the correct derivation must yield specific spans, since this appears to require knowledge of the parse itself? We have ex-

[2]We are overloading the use of β for convenience. Here, β refers to the variable ratio dependent on the particular category, whereas the β value used in supertagging is a *cutoff* applied to the variable ratio.

plored constraints derived from shallow parsing and from the raw sentence. Our results demonstrate that simple constraints can reduce parsing time significantly without loss of coverage or accuracy.

Chunk tags were used to create constraints. We experimented with both gold standard chunks from the Penn Treebank and also chunker output from the C&C chunk tagger. The tagger is very similar to the Maximum Entropy POS tagger described in Curran and Clark (2003). Only NP chunks were used because the accuracy of the tagger for other chunks is lower. The Penn Treebank chunks required modification because CCGbank analyses some constructions differently. We also created longer NPs by concatenating adjacent base NPs, for example in the case of possessives.

A number of *punctuation constraints* were used and had a significant impact especially for longer sentences. There are a number of punctuation rules in CCGbank which absorb a punctuation mark by combining it with a category and returning a category of the same type. These rules are very productive, combining with many constituent types. However, in CCGbank the sentence final punctuation is always attached at the root. A constraint on the first $n-1$ words was added to force the parser to only attach the sentence final punctuation once the rest of the sentence has been parsed.

Constraints are placed around parenthesised and quoted phrases that usually form constituents before attaching elsewhere. Constraints are also placed around phrases bound by colons, semicolons, or hyphens. These constraints are especially effective for long sentences with many clauses separated by semicolons, reducing the sentence to a number of smaller units which significantly improves parsing efficiency.

In some instances, adding constraints can be harmful to parsing efficiency and/or accuracy. Lack of precision in the constraints can come from noisy output from NLP components, e.g. the chunker, or from rules which are not always applicable, e.g. punctuation constraints. We find that the punctuation constraints are particularly effective while the gold standard chunks are required to gain any benefit for the NP constraints. Adding constraints also has the potential to increase coverage because the reduced search space means that longer sentences can be parsed without exceeding the pre-defined limits on chart size.

5 Selective Beam Search

Beam search involves greedy elimination of low probability partial derivations before they can form complete derivations. It is used in many parsers to reduce the search space, for example Collins (2003). We use a *variable width beam* where all categories c in a particular cell C that satisfy $\mathrm{score}(c) < \max\{\mathrm{score}(x)|x \in C\} - B$, for some beam cutoff B, are removed. The category scores $\mathrm{score}(c)$ are log probabilities.

In the C&C parser, the entire packed chart is constructed first and then the spanning derivations are marked. Only the partial derivations that form part of spanning derivations are scored to select the best parse, which is a small fraction of the categories in the chart. Because the categories are scored with a complex statistical model with a large number of features, the time spent calculating scores is significant. We found that applying a beam to every cell during the construction of the chart was more expensive than not using the beam at all. When the beam was made harsh enough to be worthwhile, it reduced accuracy and coverage significantly.

We propose *selective beam search* where the beam is only applied to spans of particular lengths. The shorter spans are most important to cull because there are many more of them and removing them has the largest impact in terms of reducing the search space. However, the supertagger already acts like a beam at the lexical category level and the parser model has fewer features at this level, so the beam may be more accurate for longer spans. We therefore expect the beam to be most effective for spans of intermediate length.

6 Experiments

The parser was trained on CCGbank sections 02-21 and section 00 was used for development. The performance is measured in terms of coverage, F-score and parsing time. The F-score is for labelled dependencies compared against the predicate-argument dependencies in CCGbank. The time reported includes loading the grammar and statistical model, which takes around 5 seconds, and parsing the 1913 sentences in section 00.

The failure rate (opposite of coverage) is broken down into sentences with length $\leq$ 40 and $>$ 40 because longer sentences are more difficult to parse and the C&C parser already has very high coverage on shorter sentences. There are 1784 1-40 word sentences and 129 41+ word sentences. The average length and standard deviation in the 41+ set are 50.8 and 31.5 respectively.

All experiments used gold standard POS tags. Original and REPAIR do not use constraints. The NP(GOLD) experiments use Penn Treebank gold standard NP chunks to determine an upper bound on the utility of chunk constraints. The times reported for NP(C&C) using the C&C chunker include the time to load the chunker model and run the chunker (around 1.3 seconds). PUNCT adds all of the punctuation constraints.

Finally the best system was compared against the original parser on section 23, which has 2257 sentences of length 1-40 and 153 of length 41+. The maximum length is only 65, which explains the high coverage for the 41+ section.

6.1 Chart Repair Results

The results in Table 1 show that chart repair gives an immediate 11.1% improvement in speed and a small 0.21% improvement in accuracy. 96.1% of sentences do not require chart repair because they are successfully parsed using the initial set of lexical categories supplied by the supertagger. Hence, 11% is a significant improvement for less than 4% of the sentences.

We believe the accuracy was improved (on top of the efficiency) because of the way the repair process adds new categories. Adding categories individually allows the parser to be influenced by the probabilities which the supertagger assigns, which are not directly modelled in the parser. If we were to add this information from the supertagger into the parser statistical model directly we would expect almost no accuracy difference between the original method and chart repair.

Table 2 shows the impact of different category ordering approaches for chart repair (with PUNCT constraints). The most effective approach is to use the information from the chart about the proportion of empty cells, which adds half as many categories on average as the β value and probability based approaches. All of our approaches significantly outperform randomly selecting extra categories. The CHART category ordering is used for the remaining experiments.

METHOD	secs	%	F-SCORE	CATS
RANDOM	70.2	-16.2	86.57	23.1
β VALUE	60.4	—	**86.66**	15.7
PROB	60.1	0.5	86.65	14.3
CHART	**57.2**	**5.3**	86.61	**7.0**

Table 2: Category ordering for chart repair.

6.2 Constraints Results

The results in Table 1 show that, without chart repair, using gold standard noun phrases does not improve efficiency, while using noun phrases identified by the C&C chunker decreases speed by 10.8%. They both also slightly reduce parsing accuracy. The number of times the parsing process had to be restarted with the constraints removed, was more costly than the reduction of the search space. This is unsurprising because the chunk data was not obtained from CCGbank and the chunker is not accurate enough for the constraints to improve parsing efficiency. The most frequent inconsistencies between CCGbank and chunks extracted from the Penn Treebank were fixed in a preprocessing step as explained in Section 4.1, but the less frequent constructions are still problematic.

The best results for parsing with constraints (without repair) were with *both* punctuation and gold standard noun phrase constraints, with 20.5% improvement in speed and 0.42% in coverage, but an F-score penalty of 0.3%. This demonstrates the possible efficiency gain with a perfect chunker – the corresponding results with the C&C chunker are still worse than without constraints. The best results without a decrease in accuracy use only punctuation constraints, with 10.4% increase in speed and 0.37% in coverage. The punctuation constraints also have the advantage of being simple to implement.

The best overall efficiency gain was obtained when punctuation and gold standard noun phrases were used with chart repair, with a 45.4% improvement in speed and 0.63% in coverage, and a 0.4% drop in accuracy. The best results without a drop in

METHOD	secs	%	F-SCORE	COVER	$n \leq 40$	$n > 40$
Original	88.3	—	86.54	98.85	0.392	11.63
REPAIR	78.5	11.1	**86.75**	99.01	0.336	10.08
NP(GOLD)	88.4	-0.1	86.27	99.06	0.224	10.85
NP(C&C)	97.8	-10.8	86.31	99.16	0.224	9.30
PUNCT	79.1	10.4	86.56	99.22	**0.168**	9.30
NP(GOLD) + PUNCT	69.8	20.5	86.24	99.27	**0.168**	8.53
NP(C&C) + PUNCT	97.0	-9.9	86.31	99.16	**0.168**	10.08
NP(GOLD) + REPAIR	65.0	26.4	86.04	99.37	0.224	6.20
NP(C&C) + REPAIR	77.5	12.2	86.35	99.37	0.224	6.20
PUNCT + REPAIR	57.2	35.2	86.61	99.48	**0.168**	5.43
NP(GOLD) + PUNCT + REPAIR	**48.2**	**45.4**	86.14	99.48	**0.168**	5.43
NP(C&C) + PUNCT + REPAIR	63.2	28.4	86.43	**99.53**	0.163	**3.88**

Table 1: Parsing performance on section 00 with constraints and chart repair

METHOD	secs	%	F-SCORE	COVER	$n \leq 40$	$n > 40$
Original	88.3	—	86.54	98.85	0.392	11.63
PUNCT	79.1	10.4	86.56	99.22	**0.168**	9.30
REPAIR	78.5	11.1	**86.75**	99.01	0.336	10.08
PUNCT + REPAIR	57.2	35.2	86.61	**99.48**	**0.168**	**5.43**
PUNCT + REPAIR + BEAM	**52.4**	**40.7**	86.56	**99.48**	**0.168**	**5.43**

Table 3: Best performance on Section 00

accuracy were with only punctuation constraints and chart repair, with improvements of 35.2% speed and 0.63% coverage. Coverage on both short and long sentences is improved – the best results show a 43% and 67% decrease in failure rate for sentence lengths in the ranges 1-40 and 41+ respectively.

6.3 Partial Beam Results

We found that using the selective beam on 1–2 word spans had negligible impact on speed and accuracy. Using the beam on 3–4 word spans had the most impact without accuracy penalty, improving efficiency by another $\sim$5%. Experiments with the selective beam on longer spans continued to improve efficiency, but with a much greater penalty in F-score, e.g. a further $\sim$5% at a cost of 0.5% F-score for 3–6 word spans. However, we are interested in efficiency improvements with negligible cost to accuracy.

6.4 Overall Results

Table 3 summarises the results for section 00. The chart repair and punctuation constraints individually increase parsing efficiency by around 10%. However, the most interesting result is that in combination they increase efficiency by over 35%. This is because the cost of rebuilding the chart when the constraints are incorrect has been significantly reduced by chart repair. Finally, the use of the selective beam gives modest improvement of 5.5%. The overall efficiency gain on section 00 is 40.7% with an additional 0.5% coverage, halving both the number of short and long sentences that fail to be parsed.

Table 4 shows the performance of the punctuation constraints, chart repair and selective beam system on section 23. The results are consistent with section 00, showing a 30.9% improvement in speed and 0.29% in coverage, with accuracy staying at roughly the same level. The results show a consistent 35-40% reduction in parsing time and a 40-65% reduction in parse failure rate.

7 Conclusion

We have introduced several modifications to CKY parsing for CCG that significantly increase parsing efficiency without an accuracy or coverage penalty.

METHOD	secs	%	F-SCORE	COVER	$n \leq 40$	$n > 40$
Original	91.3	—	**86.92**	99.29	0.621	1.961
PUNCT + REPAIR + BEAM	**58.7**	**35.7**	86.82	**99.58**	**0.399**	**0.654**

Table 4: Best performance on Section 23

Chart repair improves efficiency by reusing the chart from the previous parse attempts. This allows us to further tighten the parser-supertagger integration by adding one lexical category at a time until a spanning derivation is found. We have also explored several approaches to selecting which category to add next. We intend to further explore strategies for determining which category to add next when a parse fails. This includes combining chart and probability based orderings. Chart repair alone gives an 11.1% efficiency improvement.

Constraints improve efficiency by avoiding the construction of sub-derivations that will not be used. They have a significant impact on parsing speed and coverage without reducing the accuracy, provided the constraints are identified with sufficient precision. Punctuation constraints give a 10.4% improvement, but NP constraints require higher precision NP chunking than is currently available for CCGbank.

Constraints and chart repair both manipulate the chart for more efficient parsing. Adding categories one at a time using chart repair is almost a form of agenda-based parsing. We intend to explore other methods for pruning the space and agenda-based parsing, in particular A* parsing (Klein and Manning, 2003), which will allow only the most probable parts of the chart to be built, improving efficiency while still ensuring the optimal derivation is found.

When all of our modifications are used parsing speed increases by 35-40% and the failure rate decreases by 40-65%, both for sentences of length 1-40 and 41+, with a negligible accuracy penalty. The result is an even faster state-of-the-art wide-coverage CCG parser.

Acknowledgements

We would like to thank the anonymous reviewers for their feedback. James Curran was funded under ARC Discovery grants DP0453131 and DP0665973.

References

Srinivas Bangalore and Aravind Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

A. Cahill, M. Burke, R. O'Donovan, J. van Genabith, and A. Way. 2004. Long-distance dependency resolution in automatically acquired wide-coverage PCFG-based LFG approximations. In *Proceedings of the 42nd Meeting of the ACL*, pages 320–327, Barcelona, Spain.

Stephen Clark and James R. Curran. 2004a. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of COLING-04*, pages 282–288, Geneva, Switzerland.

Stephen Clark and James R. Curran. 2004b. Parsing the WSJ using CCG and log-linear models. In *Proceedings of ACL-04*, pages 104–111, Barcelona, Spain.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–637.

James R. Curran and Stephen Clark. 2003. Investigating GIS and smoothing for maximum entropy taggers. In *Proceedings of the 10th Meeting of the EACL*, pages 91–98, Budapest, Hungary.

James R. Curran, Stephen Clark, and David Vadas. 2006. Multi-tagging for lexicalized-grammar parsing. In *Proceedings of COLING/ACL-06*, pages 697–704, Sydney, Austrailia.

Julia Hockenmaier and Mark Steedman. 2002. Generative models for statistical parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Meeting of the ACL*, pages 335–342, Philadelphia, PA.

Julia Hockenmaier. 2003. *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh.

Ron Kaplan, Stefan Riezler, Tracy H. King, John T. Maxwell III, Alexander Vasserman, and Richard Crouch. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of the Human Language Technology Conference and the 4th Meeting of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL'04)*, Boston, MA.

J. Kasami. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.

Dan Klein and Christopher D. Manning. 2003. A* parsing: Fast exact Viterbi parse selection. In *Proceedings of Human Language Technology and the North American Chapter of the Association for Computational Linguistics Conference*, pages 119–126, Edmond, Canada.

Robert Malouf and Gertjan van Noord. 2004. Wide coverage parsing with stochastic attribute value grammars. In *Proceedings of the IJCNLP-04 Workshop: Beyond shallow analyses - Formalisms and statistical modeling for deep analyses*, Hainan Island, China.

Adwait Ratnaparkhi. 1996. A maximum entropy part-of-speech tagger. In *Proceedings of the EMNLP Conference*, pages 133–142, Philadelphia, PA.

Anoop Sarkar and Aravind Joshi. 2003. Tree-adjoining grammars and its application to statistical parsing. In Rens Bod, Remko Scha, and Khalil Sima'an, editors, *Data-oriented parsing*. CSLI.

Mark Steedman. 2000. *The Syntactic Process*. The MIT Press, Cambridge, MA.

Kristina Toutanova, Christopher Manning, Stuart Shieber, Dan Flickinger, and Stephan Oepen. 2002. Parse disambiguation for a rich HPSG grammar. In *Proceedings of the First Workshop on Treebanks and Linguistic Theories*, pages 253–263, Sozopol, Bulgaria.

Yoshimasa Tsuruoka and Jun'ichi Tsujii. 2004. Iterative cky parsing for probabilistic context-free grammars. In *Proceedings of the IJCNLP conference*, pages 52–60, Hainan Island, China.

D. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.

Efficiency in Unification-Based N-Best Parsing

Yi Zhang[♣], **Stephan Oepen**[♢], **and John Carroll**[♡]

[♣]Saarland University, Department of Computational Linguistics, and DFKI GmbH (Germany)
[♢]University of Oslo, Department of Informatics (Norway)
[♡]University of Sussex, Department of Informatics (UK)

Abstract

We extend a recently proposed algorithm for n-best unpacking of parse forests to deal efficiently with (a) Maximum Entropy (ME) parse selection models containing important classes of non-local features, and (b) forests produced by unification grammars containing significant proportions of globally inconsistent analyses. The new algorithm empirically exhibits a linear relationship between processing time and the number of analyses unpacked at all degrees of ME feature non-locality; in addition, compared with agenda-driven best-first parsing and exhaustive parsing with post-hoc parse selection it leads to improved parsing speed, coverage, and accuracy.[†]

1 Background—Motivation

Technology for natural language analysis using linguistically precise grammars has matured to a level of coverage and efficiency that enables parsing of large amounts of running text. Research groups working within grammatical frameworks like CCG (Clark & Curran, 2004), LFG (Riezler et al., 2002), and HPSG (Malouf & van Noord, 2004; Oepen, Flickinger, Toutanova, & Manning, 2004; Miyao, Ninomiya, & Tsujii, 2005) have successfully integrated broad-coverage computational grammars with sophisticated statistical parse selection models. The former delineate the space of possible analyses, while the latter provide a probability distribution over competing hypotheses. Parse selection approaches for these frameworks often use discriminative Maximum Entropy (ME) models, where the probability of each parse tree, given an input string, is estimated on the basis of select properties (called features) of the tree (Abney, 1997; Johnson, Geman, Canon, Chi, & Riezler, 1999). Such features, in principle, are not restricted in their domain of locality, and enable the parse selection process to take into account properties that extend beyond local contexts (i.e. sub-trees of depth one).

There is a trade-off in this set-up between the accuracy of the parse selection model, on the one hand, and the efficiency of the search for the best solution(s), on the other hand. Extending the context size of ME features, within the bounds of available training data, enables increased parse selection accuracy. However, the interplay of the core parsing algorithm and the probabilistic ranking of alternate (sub-)hypotheses becomes considerably more complex and costly when the feature size exceeds the domain of locality (of depth-one trees) that is characteristic of phrase structure grammar-based formalisms. One current line of research focuses on finding the best balance between parsing efficiency and parse selection techniques of increasing complexity, aiming to identify the most probable solution(s) with minimal effort.

This paper explores a range of techniques, combining a broad-coverage, high-efficiency HPSG parser with a series of parse selection models with varying context size of features. We sketch three general scenarios for the integration: (a) a baseline sequential configuration, where all results are enumerated first, and subsequently ranked; (b) an interleaved but approximative solution, performing a greedy search for an n-best list of results; and (c) a two-phase approach, where a complete packed for-

[†]The first author warmly acknowledges the guidance of his PhD advisors, Valia Kordoni and Hans Uszkoreit. We are grateful to Ulrich Callmeier, Berthold Crysmann, Dan Flickinger, and Erik Velldal for many discussions and their support. We thank Ron Kaplan, Martin Kay, and Bob Moore for providing insightful information about related approaches, notably the XLE and CLE parsers.

est is created and combined with a specialized graph search procedure to selectively enumerate results in (globally) correct rank order. Although conceptually simple, the second technique has not previously been evaluated for HPSG parsing (to the best of our knowledge). The last of these techniques, which we call *selective unpacking*, was first proposed by Carroll & Oepen (2005) in the context of chart-based generation. However, they only provide an account of the algorithm for local ME properties and assert that the technique should generalize to larger contexts straightforwardly. This paper describes this generalization of selective unpacking, in its application to parsing, and demonstrates that the move from features that resemble a context-free domain of locality to features of, in principle, arbitrary context size can indeed be based on the same algorithm, but the required extensions are non-trivial.

The structure of the paper is as follows. Section 2 summarizes our formalism, grammars used, parse selection approach, and training and test data. Section 3 discusses the range of possibilities for structuring the process of statistical, grammar-based parsing, and Sections 4 to 6 describe our approach to efficient n-best parsing. We present experimental results in Section 7, compare our approach to previous ones (Section 8), and finally conclude.

2 Overall Set-up

While couched in the HPSG framework, the techniques explored here are applicable to the larger class of unification-based grammar formalisms. We make use of the DELPH-IN[1] reference formalism, as implemented by a variety of systems, including the LKB (Copestake, 2002) and PET (Callmeier, 2002). For the experiments discussed here, we adapted the open-source PET parsing engine in conjunction with two publicly available grammars, the English Resource Grammar (ERG; Flickinger, 2000) and the DFKI German Grammar (GG; Müller & Kasper, 2000, Crysmann, 2005). Our parse selection models were trained and evaluated on HPSG treebanks that are distributed with these grammars. The following paragraphs summarize relevant properties of the structures manipulated by the parser,

[1]Deep Linguistic Processing with HPSG, an open-source repository of grammars and processing tools; see ‘`http://www.delph-in.net/`’.

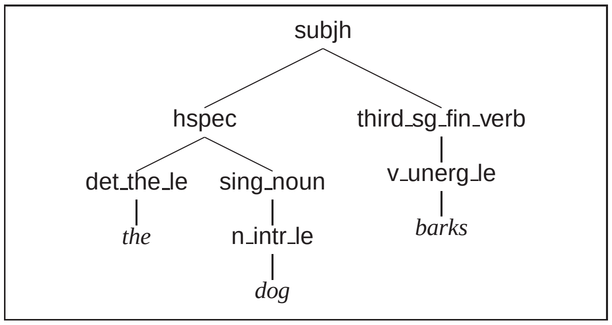


Figure 1: Sample HPSG derivation tree for the sentence *the dog barks*. Phrasal nodes are labeled with identifiers of grammar rules, and (pre-terminal) lexical nodes with class names for types of lexical entries.

followed by relevant background on parse selection.

Figure 1 shows an example ERG derivation tree. Internal tree nodes are labeled with identifiers of grammar rules, and leaves with lexical entries. The derivation tree provides complete information about the actual HPSG analysis, in the sense that it can be viewed as a recipe for computing it. Lexical entries and grammar rules alike are ultimately just feature structures, complex and highly-structured linguistic categories. When unified together in the configuration depicted by the derivation tree, the resulting feature structure yields an HPSG sign, a detailed representation of the syntactic and semantic properties of the input string. Just as the full derivation denotes a feature structure, so do its sub-trees, and for grammars like the ERG and GG each such structure will contain hundreds of feature – value pairs.

Because of the lexicalized nature of HPSG (and similar frameworks) our parsers search for well-formed derivations in a pure bottom-up fashion. Other than that, there are no hard-wired assumptions about the order of computation, i.e. the specific parsing strategy. Our basic set-up closely mimics that of Oepen & Carroll (2002), where edges indexed by sub-string positions in a chart represent the nodes of the tree, recording both a feature structure (as its category label) and the identity of the underlying lexical entry or rule in the grammar. Multiple edges derived for identical sub-strings can be ‘packed’ into a single chart entry in case their feature structures are compatible, i.e. stand in an equivalence or subsumption relation. By virtue of having each edge keep back-pointers to its daughter edges—the immediate sub-nodes in the tree whose combination resulted in the mother edge—the parse forest provides a complete and *explicit* encoding of all possible results in a maximally compact form.[2] A simple unpacking procedure is obtained from the cross-multiplication of all local combinatorics, which is directly amenable to dynamic programming.

Figure 2 shows a hypothetical forest (on the left), where sets of edges exhibiting local ambiguity have been packed into a single 'representative' edge, viz. the one in each set with one or more incoming dominance arcs. Confirming the findings of Oepen & Carroll (2002), in our experiments packing under feature structure subsumption is much more effective than packing under mere equivalence, i.e. for each pair of edges (over identical sub-strings) that stand in a subsumption relation, a technique that Oepen & Carroll (2002) termed retro-active packing ensures that the more general of the two edges remains in the chart. When packing under subsumption, however, some of the cross-product of local ambiguities in the forest may not be globally consistent. Assume for example that, in Figure 2, edges $\boxed{6}$ and $\boxed{8}$ subsume $\boxed{7}$ and $\boxed{9}$, respectively; combining $\boxed{7}$ and $\boxed{9}$ into the same tree during unpacking can in principle fail. Thus, unpacking effectively needs to deterministically replay unifications, but this extra expense in our experience is negligible when compared to the decreased cost of constructing the forest under subsumption. In Section 3 we argue that this very property, in addition to increasing parsing efficiency, interacts beneficially with parse selection and on-demand enumeration of results in rank order.

Following (Johnson et al., 1999), a conditional ME model of the probabilities of trees $\{t_1 \ldots t_n\}$ for a string s, and assuming a set of feature functions $\{f_1 \ldots f_m\}$ with corresponding weights $\{\lambda_1 \ldots \lambda_m\}$, is defined as:

$$p(t_i|s) = \frac{\exp \sum_j \lambda_j f_j(t_i)}{\sum_{k=1}^{n} \exp \sum_j \lambda_j f_j(t_k)} \qquad (1)$$

[2]This property of parse forests is not a prerequisite of the chart parsing framework. The basic CKY procedure (Kasami, 1965), for example, as well as many unification-based adaptations (e.g. the Core Language Engine; Moore & Alshawi, 1992) merely record the local category of each edge, which is sufficient for the recognition task and simplifies the search. However, reading out complete trees from the chart, then, amounts to a limited form of search, going back to the rules of the grammar itself to (re-)discover decomposition relations among chart entries.

Type	Sample Features
1	⟨0 subjh hspec third_sg_fin_verb⟩
1	⟨1 △ subjh hspec third_sg_fin_verb⟩
1	⟨0 hspec det_the_le sing_noun⟩
1	⟨1 subjh hspec det_the_le sing_noun⟩
1	⟨2 △ subjh hspec det_the_le sing_noun⟩
2	⟨0 subjh third_sg_fin_verb⟩
2	⟨0 subjh hspce⟩
2	⟨1 subjh hspec det_the_le⟩
2	⟨1 subjh hspec sing_noun⟩
3	⟨1 n_intr_le *dog*⟩
3	⟨2 det_the_le n_intr_le *dog*⟩
3	⟨3 ◁ det_the_le n_intr_le *dog*⟩

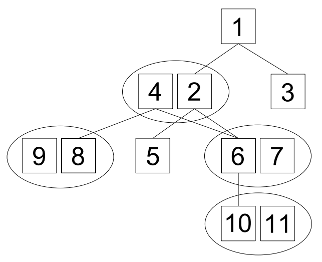
Table 1: Examples of structural features extracted from the derivation tree in Figure 1. The *Type* column indicates the template corresponding to each sample feature; the integer that starts each feature indicates the degree of grandparenting (in the case of type 1 and 2 features) or n-gram size (type 3 features). The symbols △ and ◁ denote the root of the tree and left periphery of the yield, respectively.

Feature functions f_j can test for arbitrary structural properties of analyses t_i, and their value typically is the number of times a specific property is present in t_i. Toutanova, Manning, Flickinger, & Oepen (2005) propose an inventory of features that perform well in HPSG parse selection; currently we restrict ourselves to the best-performing of these, of the form illustrated in Table 1, comprising depth-one sub-trees (or portions of these) with grammar-internal identifiers as node labels, plus optionally a chain of one or more dominating nodes (i.e. levels of grandparents). If a grandparents chain is present then the feature is non-local. For expository purposes, Table 1 includes another feature type, n-grams over leaf nodes of the derivation; in Section 5 below we speculate about the incorporation of these (and similar) features in our algorithm.

3 Interleaving Parsing and Ranking

At an abstract level, given a grammar and an associated ME parse selection model, there are three basic ways of combining them in order to find the single 'best' or small set of n-best results.

The first way is a naïve sequential set-up, in which the parser first enumerates the full set of analyses, computes a score for each using the model, and returns the highest-ranking n results. For carefully



$$\boxed{1} \rightarrow \left\langle \boxed{2}\boxed{3} \right\rangle \mid \left\langle \boxed{4}\boxed{3} \right\rangle$$
$$\boxed{2} \rightarrow \left\langle \boxed{5}\boxed{6} \right\rangle \mid \left\langle \boxed{5}\boxed{7} \right\rangle$$
$$\boxed{4} \rightarrow \left\langle \boxed{8}\boxed{6} \right\rangle \mid \left\langle \boxed{8}\boxed{7} \right\rangle \mid \left\langle \boxed{9}\boxed{6} \right\rangle \mid \left\langle \boxed{9}\boxed{7} \right\rangle$$
$$\boxed{6} \rightarrow \left\langle \boxed{10} \right\rangle \mid \left\langle \boxed{11} \right\rangle$$

Figure 2: Sample forest and sub-node decompositions: ovals in the forest (on the left) indicate packing of edges under subsumption, i.e. edges $\boxed{4}$, $\boxed{7}$, $\boxed{9}$, and $\boxed{11}$ are *not* in the chart proper. During unpacking, there will be multiple ways of instantiating a chart edge, each obtained from cross-multiplying alternate daughter sequences locally. The elements of this cross-product we call *decomposition*, and they are pivotal points both for stochastic scoring and dynamic programming in selective unpacking. The table on the right shows all non-leaf decompositions for our example packed forest: given two ways of decomposing $\boxed{6}$, there will be three candidate ways of instantiating $\boxed{2}$ and six for $\boxed{4}$, respectively, for a total of nine full trees.

crafted grammars and inputs of average complexity the approach can perform reasonably well.

Another mode of operation is to organize the parser's search according to an agenda (i.e. priority queue) that assigns numeric scores to parsing moves (Erbach, 1991). Each such move is an application of the fundamental rule of chart parsing, combining an active and a passive edge, and the scores represent the expected 'figure of merit' (Caraballo & Charniak, 1998) of the resulting structure. Assuming a parse selection model of the type sketched in Section 2, we can determine the agenda priority for a parsing move according to the (unnormalized) ME score of the derivation (sub-)tree that would result from its successful execution. Note that, unlike in probabilistic context-free grammars (PCFGs), ME scores of partial trees do not necessarily decrease as the tree size increases; instead, the distribution of feature weights is in the range $(-\infty, +\infty)$, centered around 0, where negative weights intuitively correspond to dis-preferred properties.

This lack of monotonicity in the scores associated with sub-trees, on the one hand, is beneficial, in that performing a greedy best-first search becomes practical: in contrast, with PCFGs and their monotonically decreasing probabilities on larger sub-trees, once the parser finds the first full tree the chart necessarily has been instantiated almost completely. On the other hand, the same property prohibits the application of exact best-first techniques like A* search, because there is no reliable future cost estimate; in this respect, our set-up differs fundamentally from that of Klein & Manning (2003) and related PCFG parsing work. Using the unnormalized sum of ME weights on a partial solution as its agenda score, effectively, means that sub-trees with low scores 'sink' to the bottom of the agenda; highly-ranked partial constituents, in turn, instigate the immediate creation of larger structures, and ideally the bottom-up agenda-driven search will greedily steer the parser towards full analyses with high scores. Given its heuristic nature, this procedure cannot guarantee that its n-best list of results corresponds to the globally correct rank order, but it may in practice come reasonably close to it. While conceptually simple, greedy best-first search does not combine easily with ambiguity packing in the chart: (a) at least when packing under subsumption, it is not obvious how to accurately compute the agenda score of packed nodes, and (b) to the extent that the greedy search avoids exploration of dis-preferred local ambiguity, the need for packing should be greatly reduced. Unfortunately, in scoring bottom-up parsing moves, ME features involving grandparenting are not applicable, leading to a second potential source of reduced parse selection accuracy. In Section 7 below, we provide an empirical evaluation of both the naïve sequential and greedy best-first approaches.

4 Selective Unpacking

Carroll & Oepen (2005) observe that, at least for grammars like the ERG, the construction of the parse forest can be very efficient (with observed polynomial complexity), especially when packing edges under subsumption. Their selective unpacking procedure, originally proposed for the forest created by a chart *generator*, aims to unpack the n-best set

```
procedure selectively-unpack-edge(edge, n) ≡
  results ← ⟨ ⟩; i ← 0;
  do
    hypothesis ← hypothesize-edge(edge, i); i ← i + 1;
    if (new ← instantiate-hypothesis(hypothesis)) then
      n ← n − 1; results ← results ⊕ ⟨new⟩;
  while (hypothesis and n ≥ 1)
  return results;

procedure hypothesize-edge(edge, i) ≡
  if (edge.hypotheses[i]) return edge.hypotheses[i];
  if (i = 0) then
    for each (decomposition in decompose-edge(edge)) do
      daughters ← ⟨ ⟩; indices ← ⟨ ⟩
      for each (edge in decomposition.rhs) do
        daughters ← daughters ⊕ ⟨hypothesize-edge(edge, 0)⟩;
        indices ← indices ⊕ ⟨0⟩;
      new-hypothesis(edge, decomposition, daughters, indices);
  if (hypothesis ← edge.agenda.pop()) then
    for each (indices in advance-indices(hypothesis.indices)) do
      if (indices ∈ hypothesis.decomposition.indices) then continue
      daughters ← ⟨ ⟩;
      for each (edge in hypothesis.decomposition.rhs) each (i in indices) do
        daughter ← hypothesize-edge(edge, i);
        if (not daughter) then daughters ← ⟨ ⟩; break
        daughters ← daughters ⊕ ⟨daughter⟩;
      if (daughters) then new-hypothesis(edge, hypothesis.decomposition, daughters, indices)
    edge.hypotheses[i] ← hypothesis;
    return hypothesis;

procedure new-hypothesis(edge, decomposition, daughters, indices) ≡
  hypothesis ← new hypothesis(decomposition, daughters, indices);
  edge.agenda.insert(score-hypothesis(hypothesis), hypothesis);
  decomposition.indices ← decomposition.indices ∪ {indices};
```

Figure 3: Selective unpacking procedure, enumerating the n best realizations for a top-level result *edge* from a packed forest. An auxiliary function decompose-edge() performs local cross-multiplication as shown in the examples in Figure 2. Another utility function not shown in pseudo-code is advance-indices(), a 'driver' routine searching for alternate instantiations of daughter edges, e.g. advance-indices($\langle 0\ 2\ 1\rangle$) $\rightarrow \{\langle 1\ 2\ 1\rangle\ \langle 0\ 3\ 1\rangle\ \langle 0\ 2\ 2\rangle\}$. Finally, instantiate-hypothesis() is the function that actually builds result trees, replaying the unifications of constructions from the grammar (as identified by chart edges) with the feature structures of daughter constituents.

of full trees from the forest, guaranteeing the globally correct rank order according to the probability distribution, with a minimal amount of search. The basic algorithm is a specialized graph search through the forest, with local contexts of optimization corresponding to packed nodes.

Each such node represents local combinatorics, and two key notions in the selective unpacking procedure are the concepts of (a) *decomposing* an edge locally into candidate ways of instantiating it, and of (b) nested contexts of local search for ranked *hypotheses* (i.e. uninstantiated edges) about candidate subtrees. See Figure 2 for examples of the decomposition of edges. Given one decomposition—i.e. a vector of candidate daughters for a particular rule—there can be multiple ways of instantiating each daughter: a parallel index vector $\vec{I} = \langle i_0 \ldots i_n\rangle$ serves to keep track of 'vertical' search among daughter hypotheses, where each index i_j denotes the i-th best instantiation (hypothesis) of the daughter at position j. If we restrict ME features to a depth of one (i.e. without grandparenting), then given the additive nature of ME scores on complete derivations, it can be guaranteed that hypothesized trees including an edge e as an immediate daughter must use the best instantiation of e in their own best instantiation. Assuming a binary rule, the corresponding hypothesis would use daughter indices of $\langle 0\,0\rangle$. The second-best instantiation, in turn, can be obtained from moving to the second-best hypothesis for *one* of the elements in the (right-hand side of the) decomposition, e.g. indices

$\langle 0\ 1 \rangle$ or $\langle 1\ 0 \rangle$ in the binary example. Hypotheses are associated with ME scores and ordered within each nested context by means of a local priority queue (stored in the original representative edge, for convenience). Therefore, nested local optimizations result in a top-down, breadth-first, exact n-best search through the packed forest, while avoiding exhaustive cross-multiplication of packed nodes.

Figure 3 shows the unchanged pseudo-code of Carroll & Oepen (2005). The main function hypothesize-edge() controls both the 'horizontal' and 'vertical' search, initializing the set of decompositions and pushing initial hypotheses onto the local agenda when called on an edge for the first time (lines 11 – 17). For each call, the procedure retrieves the current next-best hypothesis from the agenda (line 18), generates new hypotheses by advancing daughter indices (while skipping over configurations seen earlier) and calling itself recursively for each new index (lines 19 – 26), and, finally, arranging for the resulting hypothesis to be cached for later invocations on the same *edge* and *i* values (line 27). Note that unification (in instantiate-hypothesis()) is only invoked on complete, top-level hypotheses, as our structural ME features can actually be evaluated *prior* to building each full feature structure. However, as Carroll & Oepen (2005) suggest, the procedure could be adapted to perform instantiation of sub-hypotheses within each local search, should additional features require it. For better efficiency, the instantiate-hypothesis() routine applies dynamic programming (i.e. memoization) to intermediate results.

5 Generalizing the Algorithm

Carroll & Oepen (2005) offer no solution for selective unpacking with larger context ME features. Yet, both Toutanova et al. (2005) and our own experiments (described in Section 7 below) suggest that properties of larger contexts and especially grandparenting can greatly improve parse selection accuracy. The following paragraphs outline how to generalize the basic selective unpacking procedure, while retaining its key properties: exact n-best enumeration with minimal search. Our generalization of the algorithm distinguishes between 'upward' contexts, with grandparenting with dominating nodes as a representative feature type, and 'downward' extensions, which we discuss for the example of lexical n-gram features.

A naïve approach to selective unpacking with grandparenting might be extending the cross-multiplication of local ambiguity to trees of more than depth one. However, with multiple levels of grandparenting this approach would greatly increase the combinatorics to be explored, and it would pose the puzzle of overlapping local contexts of optimization. Choices made among the alternates for one packed node would interact with other ambiguity contexts in their internal nodes, rather than merely at the leaves of their decompositions. However, it is sufficient to keep the depth of decompositions to minimal sub-trees and rather contextualize each decomposition as a whole. Assuming our sample forest and set of decompositions from Figure 2, let $\langle \boxed{1}\boxed{4} \rangle : \boxed{6} \rightarrow \langle \boxed{10} \rangle$ denote the decomposition of node $\boxed{6}$ in the context of $\boxed{4}$ and $\boxed{1}$ as its immediate parents. When descending through the forest, hypothesize-edge() can, without significant extra cost, maintain a vector $\vec{P} = \langle p_n \ \ldots \ p_0 \rangle$ of parents of the current node, for n-level grandparenting. For each packed node, the bookkeeping elements of the graph search procedure need to be contextualized on $\vec{P}$, viz. (a) the edge-local priority queue, (b) the record of index vectors hypothesized already, and (c) the cache of previous instantiations. Assuming each is stored in an associative array, then all references to edge.agenda in the original procedure can be replaced by edge.agenda[$\vec{P}$], and likewise for other slots. With these extensions in place, the original control structure of nested, on-demand creation of hypotheses and dynamic programming of partial results can be retained, and for each packed node with multiple parents ($\boxed{6}$ in our sample forest) there will be parallel, contextualized partitions of optimization. Thus, extra combinatorics introduced in this generalized procedure are confined to only such nodes, which (intuitively at least) appears to establish the lower bound of added search needed—while keeping the algorithm non-approximative. Section 7 provides empirical data on the degradation of the procedure in growing levels of grandparenting and the number of n-best results to be extracted from the forest.

Finally, we turn to enlarged feature contexts that capture information from nodes *below* the elements of a local decomposition. Consider the example of feature type 3 in Table 1, n-grams (of various size) over properties of the yield of the parse tree. For now we only consider lexical *b*i-grams. For an edge e dominating a sub-string of n words $\langle w_i \ \ldots \ w_{i+n-1}\rangle$ there will be $n-1$ bi-grams internal to e, and two bi-grams that interact with w_{i-1} and w_{i+n}—which will be determined by the left- and right-adjacent edges to e in a complete tree. The internal bi-grams are unproblematic, and we can assume that ME weights corresponding to these features have been included in the sum of weights associated to e. Seeing that e may occur in multiple trees, with different sister edges, the selective unpacking procedure has to take this variation into account when evaluating local contexts of optimization.

Let ${}_xe_y$ denote an edge e, with x and y as the lexical types of its leftmost and rightmost daughters, respectively. Returning to our sample forest, assume lexicalizations ${}_\beta\boxed{10}_\beta$ and ${}_\gamma\boxed{11}_\gamma$ (each spanning only one word), with $\beta \neq \gamma$. Obviously, when decomposing $\boxed{4}$ as $\langle\boxed{8}\boxed{6}\rangle$, its ME score, in turn, will depend on the choice made in the expansion of $\boxed{6}$: the sequences $\left\langle {}_\alpha\boxed{8}_\alpha \, {}_\beta\boxed{6}_\beta \right\rangle$ and $\left\langle {}_\alpha\boxed{8}_\alpha \, {}_\gamma\boxed{6}_\gamma \right\rangle$ will differ in (at least) the scores associated with the bigrams $\langle \alpha \, \beta \rangle$ vs. $\langle \alpha \, \gamma \rangle$. Accordingly, when evaluating candidate decompositions of $\boxed{4}$, the number of hypotheses that need to be considered is doubled; as an immediate consequence, there can be up to eight distinct lexicalized variants for the decomposition $\boxed{1} \rightarrow \langle\boxed{4}\boxed{3}\rangle$ further up in the tree. It may look as if combinatorics will cross-multiply throughout the tree—in the worst case returning us to an exponential number of hypotheses—but this is fortunately not the case: regarding the external bi-grams of $\boxed{1}$, node $\boxed{6}$ no longer participates in its left- or rightmost periphery, so variation internal to $\boxed{6}$ is not a multiplicative factor at this level. This is essentially the observation of Langkilde (2000), and her bottom-up factoring of n-gram computation is easily incorporated into our top-down selective unpacking control structure. At the point where hypothesize-edge() invokes itself recursively (line 23 in Figure 3), its return value is now a set of lexicalized alternates, and hypothesis creation (in line 26) can take into account the local cross-product of all such alternation.

Including additional properties from non-local subtrees (for example higher-order n-grams and head lexicalization) is a straightforward extension of this scheme, replacing our per-edge left- and rightmost periphery symbols with a generalized vector of externally relevant, internal properties. In addition to traditional (head) lexicalization as we have just discussed it, such extended 'downward' properties on decompositions—percolated from daughters to mothers and cross-multiplied as appropriate—could include metrics of constituent weight too, for example to enable the ME model to prefer 'balanced' coordination structures.

However, given that Toutanova et al. (2005) obtain only marginally improved parse selection accuracy from the inclusion of n-gram (and other lexical) ME features, we have left the implementation of lexicalization and empirical evaluation for future work.

6 Failure Caching and Propagation

As we pointed out at the end of Section 4, during the unpacking phase, unification is only replayed in instantiate-hypothesis() on the top-level hypotheses. It is only at this step that inconsistencies in the local combinatorics are discovered. However, such a discovery can be used to improve the unpacking routine by (a) avoiding further unification on hypotheses that have already failed to instantiate, (b) avoiding creating new hypotheses based on failed sub-hypotheses. This requires some changes to the routines instantiate-hypothesis() and hypothesize-edge(), as well as an extra boolean marker for each hypothesis.

The extended instantiate-hypothesis() starts by checking whether the hypothesis is already marked as failed. If it is not so marked, the routine recursively instantiates all sub-hypotheses. Any failure will again lead to instant return. Otherwise, unification is used to create a new edge from the outcome of the sub-hypothesis instantiations. If this unification fails, the current hypothesis is marked. Moreover, all its ancestor hypotheses are also marked (by recursively following the pointers to the direct parent hypotheses) as they are also guaranteed to fail.

Correspondingly, hypothesize-edge() needs to check the instantiation failure marker to avoid returning hypotheses that are guaranteed to fail. If a hypothesis coming out of the agenda is already marked as failed, it will be used to create new hypotheses (with advance-indices()), but dropped afterward. Subsequent hypotheses will be popped from the agenda until either a hypothesis that is not marked as failed is returned, or the agenda is empty.

Moreover, hypothesize-edge() also needs to avoid creating new hypotheses based on failed sub-hypotheses. When a failed sub-hypothesis is found, the creation of the new hypothesis is skipped. But the index vector $\vec{I}$ may not be simply discarded. Otherwise hypotheses based on advance-indices($\vec{I}$) will not be reachable in the search. On the other hand, simply adding every advance-indices($\vec{I}$) on to the pending creation list is not efficient either in the case where multiple sub-hypotheses fail.

To solve the problem, we compute a failure vector $\vec{F} = \langle f_0 \ldots f_n \rangle$, where f_j is 1 when the sub-hypothesis at position j is known as failed, and 0 otherwise. If a sub-hypothesis at position j is failed then all the index vectors having value i_j at position j must also fail. By putting the result of $\vec{I} + \vec{F}$ on the pending creation list, we can safely skip the failed rows of sub-hypotheses, while not losing the reachability of the others. As an example, suppose we have a ternary index vector $\langle 3\ 1\ 2 \rangle$ for which a new hypothesis is to be created. By checking the instantiation failure marker of the sub-hypotheses, we find that the first and the third sub-hypotheses are already marked. The failure recording vector will then be $\langle 1\ 0\ 1 \rangle$. By putting $\langle 4\ 1\ 3 \rangle = \langle 3\ 1\ 2 \rangle + \langle 1\ 0\ 1 \rangle$ on to the pending hypothesis creation list, the failed sub-hypotheses are skipped.

We evaluate the effects of instantiation failure caching and propagation below in Section 7.

7 Empirical Results

To evaluate the performance of the selective unpacking algorithm, we carried out a series of empirical evaluations with the ERG and GG, in combination with a modified version of the PET parser. When running the ERG we used as our test set the *JH4* section of the LOGON treebank[3], which contains 1603 items with an average sentence length of 14.6 words. The remaining LOGON treebank (of around 8,000 items) was used in training the various ME parse disambiguation models. For the experiment with GG, we designated a 2825-item portion of the DFKI Verb*mobil* treebank[4] for our tests, and trained ME models on the remaining 10,000 utterances. At only 7.4 words, the average sentence length is much shorter in the Verb*mobil* data.

We ran seven different configurations of the parser with different search strategies and (un-)packing mechanisms:

- Agenda driven greedy n-best parsing using the ME score without grandparenting features; no local ambiguity packing;
- Local ambiguity packing with exhaustive unpacking, without grandparenting features;

Configuration	GP	Coverage	Time (s)
greedy best-first	0	91.6%	3889
exhaustive unpacking	0	84.5%	4673
selective unpacking	0	94.3%	2245
	1	94.3%	2529
	2	94.3%	3964
	3	94.2%	3199
	4	94.2%	3502

Table 2: Coverage on the ERG for different configurations, with fixed resource consumption limits (of 100k passive edges or 300 seconds). In all cases, up to ten 'best' results were searched, and *Coverage* shows the percentage of inputs that succeed to parse within the available resource. *Time* shows the end-to-end processing time for each batch.

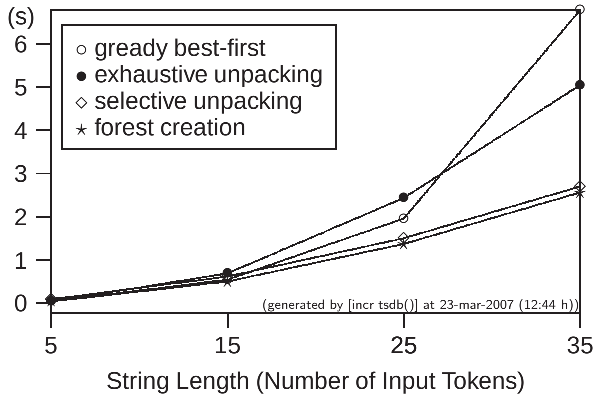


Figure 4: Parsing times for different configurations using the ERG, in all three cases searching for up to ten results, without the use of grandparenting.

[3]The treebank is comprised of several booklets of edited, instructional texts on backcountry activities in Norway. The data is available from the LOGON web site at '`http://www.emmtee.net`'.

[4]The data in this treebank is taken from transcribed appointment scheduling dialogues; see '`http://gg.dfki.de/`' for further information on GG and its treebank.

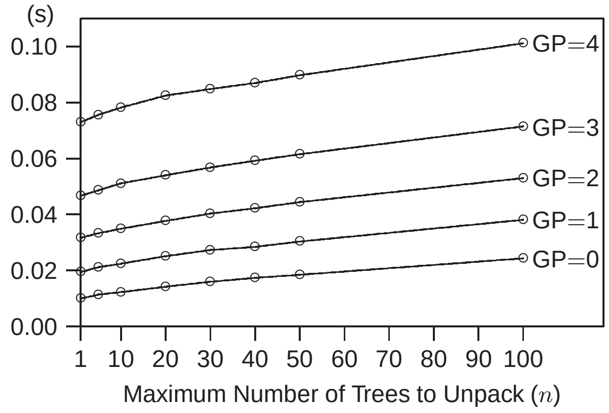


Figure 5: Mean times for selective unpacking of all test items for n-best parsing with the ERG, for varying n and grandparenting (GP) levels

Configuration	Exact Match	Top Ten
random choice	11.34	43.06
no grandparenting	52.52	68.38
greedy best-first	51.79	69.48
grandparenting[1]	56.83	85.33
grandparenting[2]	56.55	84.14
grandparenting[3]	56.37	84.14
grandparenting[4]	56.28	84.51

Table 3: Parse selection accuracy for various levels of grandparenting. The *exact match* column shows the percentage of cases in which the correct tree, according to the treebank, was ranked highest by the model; conversely, the *top ten* column indicates how often the correct tree was among the ten top-ranking results.

- Local ambiguity packing and selective unpacking for n-best parsing, with 0 through 4 levels of grandparenting (GP) features.

As a side-effect of differences in efficiency, some configurations could not complete parsing all sentences given reasonable memory constraints (which we set at a limit of 100k passive edges or 300 seconds processing time per item). The overall coverage and processing time of different configurations on *JH4* are given in Table 2.

The correlation between processing time and coverage is interesting. However, it makes the efficiency comparison difficult as parser behavior is not clearly defined when the memory limit is exceeded. To circumvent this problem, in the following experiments we average only over those 1362 utterances from *JH4* that complete parsing within the resource limit in all seven configurations. Nevertheless, it must be noted that this restriction potentially reduces efficiency differences between configurations, as some of the more challenging inputs (which typically lead to the largest differences) are excluded.

Figure 4 compares the processing time of different configurations. The difference is much more significant for longer sentences (i.e. with more than 15 words). If the parser unpacks exhaustively, the time for unpacking grows with sentence length at a quickly increasing rate. In such cases, the efficiency gain with ambiguity packing in the parsing phase is mostly lost in the unpacking phase. The graph shows that greedy best-first parsing without packing outperforms exhaustive unpacking for sentences of less than 25 words. With sentences longer than 25 words, the packing mechanism helps the parser to overtake greedy best-first parsing, although the exhaustive unpacking time also grows fast.

With the selective unpacking algorithm presented in the previous sections, unpacking time is reduced, and grows only slowly as sentence length increases. Unpacking up to ten results, when contrasted with the timings for forest creation (i.e. the first parsing phase) in Figure 4, adds a near-negligible extra cost to the total time required for both phases. Moreover, Figure 5 shows that with selective unpacking, as n is increased, unpacking time grows roughly linearly for all levels of grandparenting (albeit always with an initial delay in unpacking the first result).

Table 4 summarizes a number of internal parser measurements using the ERG with different packing/unpacking settings. Besides the difference in processing time, we also see a significant difference in *“space”* between exhaustive and selective unpacking. Also, the difference in *“unifications”* and *“copies”* indicates that with our selective unpacking algorithm, these expensive operations on typed feature structures are significantly reduced.

In return for increased processing time (and marginal loss in coverage) when using grandparenting features, Table 3 shows some large improvements in parse selection accuracy (although the picture is less clear-cut at higher-order levels of grandparenting[5]). A balance point between efficiency

[5]The models were trained using the open-source TADM package (Malouf, 2002), using default hyper-parameters for all configurations, viz. a convergence threshold of 10^{-8}, variance of the prior of 10^{-4}, and frequency cut-off of 5. It is likely that

	Configuration	GP	Unifications (#)	Copies (#)	Space (kbyte)	Unpack (s)	Total (s)
≤ 15 words	greedy best-first	0	1845	527	2328	–	0.12
	exhaustive unpacking	0	2287	795	8907	0.01	0.12
	selective unpacking	0	1912	589	8109	0.00	0.12
		1	1913	589	8109	0.01	0.12
		2	1914	589	8109	0.01	0.12
		3	1914	589	8110	0.01	0.12
		4	1914	589	8110	0.02	0.13
> 15 words	greedy best-first	0	25233	5602	24646	–	1.66
	exhaustive unpacking	0	39095	15685	80832	0.85	1.95
	selective unpacking	0	17489	4422	33326	0.03	1.17
		1	17493	4421	33318	0.05	1.21
		2	17493	4421	33318	0.09	1.25
		3	17495	4422	33321	0.13	1.27
		4	17495	4422	33320	0.21	1.34

Table 4: Contrasting the efficiency of various (un-)packing settings in use with ERG on short (top) and medium-length (bottom) inputs; in each configuration, up to ten trees are extracted. *Unification* and *Copies* is the count of top-level FS operations, where only successful unifications require a subsequent copy (when creating a new edge). *Unpack* and *Total* are unpacking and total parse time, respectively.

and accuracy can be made according to application needs.

Finally, we compare the processing time of the selective unpacking algorithm with and without instantiation failure caching and propagation (as described in Section 4 above). The empirical results for GG are summarized in Table 5, showing clearly that the technique reduced unnecessary hypotheses and instantiation failures. The design philosophy of the ERG and GG differ. During the first, forest creation phase, GG suppresses a number of features (in the HPSG sense, not the ME sense) that can actually constrain the combinatorics of edges. This move makes the packed forest more compact, but it implies that unification failures will be more frequent during unpacking. In a sense, GG thus moves part of the search for globally consistent derivations into the second phase, and it is possible for the forest to contain 'result' trees that ultimately turn out to be incoherent. Dynamic programming of instantiation failures makes this approach tractable, while retaining the general breadth-first characteristic of the selective unpacking regime.

further optimization of hyper-parameters for individual configurations would moderately improve model performance, especially for higher-order grandparenting levels with large numbers of features.

8 Discussion

The approach to n-best parsing described in this paper takes as its point of departure recent work of Carroll & Oepen (2005), which describes an efficient algorithm for unpacking n-best trees from a forest produced by a chart-based sentence generator and containing local ME properties with associated weights. In an almost contemporaneous study, but in the context of parsing with treebank grammars, Huang & Chiang (2005) develop a series of increasingly efficient algorithms for unpacking n-best results from a weighted hypergraph representing a parse forest. The algorithm of Carroll & Oepen (2005) and the final one of Huang & Chiang (2005) are essentially equivalent, and turn out to be reformulations of an approach originally described by Jiménez & Marzal (2000) (although expressed there only for grammars in Chomsky Normal Form).

In this paper we have considered ME properties that extend beyond immediate dominance relations, extending up to 4 levels of grandparenting. Previous work has either assumed properties that are restricted to the minimal parse fragments (i.e. subtrees of depth one) that make up the packed representation (Geman & Johnson, 2002), or has taken a more relaxed approach by allowing non-local prop-

Configuration	**Unifications** (#)	**Copies** (#)	**Hypotheses** (#)	**Space** (kbyte)	**Unpack** (ms)	**Total** (ms)
greedy best-first	5980	1447	–	9202	–	400
selective, no caching	5535	1523	1245	27188	70	410
selective, with cache	4915	1522	382	27176	10	350

Table 5: Efficiency effects of the instantiation failure caching and propagation with GG, without grandparenting. All statistics are averages over the 1941 items that complete within the resource bounds in all three configurations. *Unification*, *Copies*, *Unpack*, and *Total* have the same interpretation as in Table 4, and *Hypotheses* is the average count of hypothesized sub-trees.

erties but without addressing the problem of how to efficiently extract the top-ranked trees from a packed forest (Miyao & Tsujii, 2002).

Probably the work closest in spirit to our approach is that of Malouf & van Noord (2004), who use an HPSG grammar comparable to the ERG and GG, non-local ME features, and a two-phase parse forest creation and unpacking approach. However, their unpacking phase uses a beam search to find a good (single) candidate for the best parse; in contrast—for ME models containing the types of non-local features that are most important for accurate parse selection—we avoid an approximative search and *efficiently* identify *exactly* the n-best parses.

When parsing with context free grammars, a (single) parse can be retrieved from a parse forest in time linear in the length of the input string (Billot & Lang, 1989). However, as discussed in Section 2, when parsing with a unification-based grammar and packing under feature structure subsumption, the cross-product of some local ambiguities may not be globally consistent. This means that additional unifications are required at unpacking time. In principle, when parsing with a pathological grammar with a high rate of failure, extracting a single consistent parse from the forest could take exponential time (see Lang (1994) for a discussion of this issue with respect to Indexed Grammars). In the case of GG, a high rate of unification failure in unpacking is dramatically reduced by our instantiation failure caching and propagation mechanism.

9 Conclusions and Future Work

We have described and evaluated an algorithm for efficiently computing the n-best analyses from a parse forest produced by a unification grammar, with respect to a Maximum Entropy (ME) model containing two classes of non-local features. The algorithm is efficient in that it empirically exhibits a linear relationship between processing time and the number of analyses unpacked, at all degrees of ME feature non-locality. It improves over previous work in providing the only exact procedure for retrieving n-best analyses from a packed forest that can deal with features with extended domains of locality and with forests created under subsumption. Our algorithm applies dynamic programming to intermediate results and local failures in unpacking alike.

The experiments compared the new algorithm with baseline systems representing other possible approaches to parsing with ME models: (a) a single phase of agenda-driven parsing with on-line pruning based on intermediate ME scores, and (b) two-phase parsing with exhaustive unpacking and post-hoc ranking of complete trees. The new approach showed better speed, coverage, and accuracy than the baselines.

Although we have dealt with the non-local ME features that in previous work have been found to be the most important for parse selection (i.e. grandparenting and n-grams), this does not exhaust the full range of features that could possibly be useful. For example, it may be the case that accurately resolving some kinds of ambiguities can only be done with reference to particular parts—or combinations of parts—of the HPSG feature structures representing the analysis of a complete constituent. To deal with such cases we are currently designing an extension to the algorithms described here which would add a ‘controlled’ beam search, in which the size of the beam was limited by the interval of score adjustments for ME features that could only be evaluated once the full linguistic structure became available. This approach would involve a constrained amount of extra search, but would still produce the exact n-best trees.

References

Abney, S. P. (1997). Stochastic attribute-value grammars. *Computational Linguistics*, *23*, 597 – 618.

Billot, S., & Lang, B. (1989). The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th Meeting of the Association for Computational Linguistics* (pp. 143 – 151). Vancouver, BC.

Callmeier, U. (2002). Preprocessing and encoding techniques in PET. In S. Oepen, D. Flickinger, J. Tsujii, & H. Uszkoreit (Eds.), *Collaborative language engineering. A case study in efficient grammar-based processing.* Stanford, CA: CSLI Publications.

Caraballo, S. A., & Charniak, E. (1998). New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, *24*(2), 275 – 298.

Carroll, J., & Oepen, S. (2005). High-efficiency realization for a wide-coverage unification grammar. In R. Dale & K. F. Wong (Eds.), *Proceedings of the 2nd International Joint Conference on Natural Language Processing* (Vol. 3651, pp. 165 – 176). Jeju, Korea: Springer.

Clark, S., & Curran, J. R. (2004). Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics* (pp. 104 – 111). Barcelona, Spain.

Copestake, A. (2002). *Implementing typed feature structure grammars.* Stanford, CA: CSLI Publications.

Crysmann, B. (2005). Relative clause extraposition in German. An efficient and portable implementation. *Research on Language and Computation*, *3*(1), 61 – 82.

Erbach, G. (1991). A flexible parser for a linguistic development environment. In O. Herzog & C.-R. Rollinger (Eds.), *Text understanding in LILOG* (pp. 74 – 87). Berlin, Germany: Springer.

Flickinger, D. (2000). On building a more efficient grammar by exploiting types. *Natural Language Engineering*, *6 (1)*, 15 – 28.

Geman, S., & Johnson, M. (2002). Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of the 40th Meeting of the Association for Computational Linguistics.* Philadelphia, PA.

Huang, L., & Chiang, D. (2005). Better k-best parsing. In *Proceedings of the 9th International Workshop on Parsing Technologies* (pp. 53 – 64). Vancouver, Canada.

Jiménez, V. M., & Marzal, A. (2000). Computation of the n best parse trees for weighted and stochastic context-free grammars. In *Proceedings of the Joint International Workshops on Advances in Pattern Recognition* (pp. 183 – 192). London, UK: Springer-Verlag.

Johnson, M., Geman, S., Canon, S., Chi, Z., & Riezler, S. (1999). Estimators for stochastic 'unification-based' grammars. In *Proceedings of the 37th Meeting of the Association for Computational Linguistics* (pp. 535 – 541). College Park, MD.

Kasami, T. (1965). *An efficient recognition and syntax algorithm for context-free languages* (Technical Report # 65-758). Bedford, MA: Air Force Cambrige Research Laboratory.

Klein, D., & Manning, C. D. (2003). A* parsing. Fast exact Viterbi parse selection. In *Proceedings of the 4th Conference of the North American Chapter of the ACL.* Edmonton, Canada.

Lang, B. (1994). Recognition can be harder than parsing. *Computational Intelligence*, *10*(4), 486 – 494.

Langkilde, I. (2000). Forest-based statistical sentence generation. In *Proceedings of the 1st Conference of the North American Chapter of the ACL.* Seattle, WA.

Malouf, R. (2002). A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the 6th Conference on Natural Language Learning.* Taipei, Taiwan.

Malouf, R., & van Noord, G. (2004). Wide coverage parsing with stochastic attribute value grammars. In *Proceedings of the IJCNLP workshop Beyond Shallow Analysis.* Hainan, China.

Miyao, Y., Ninomiya, T., & Tsujii, J. (2005). Corpus-oriented grammar development for acquiring a Head-Driven Phrase Structure Grammar from the Penn Treebank. In K.-Y. Su, J. Tsujii, J.-H. Lee, & O. Y. Kwong (Eds.), *Natural language processing* (Vol. 3248, pp. 684 – 693). Hainan Island, China.

Miyao, Y., & Tsujii, J. (2002). Maximum entropy estimation for feature forests. In *Proceedings of the Human Language Technology Conference.* San Diego, CA.

Moore, R. C., & Alshawi, H. (1992). Syntactic and semantic processing. In H. Alshawi (Ed.), *The Core Language Engine* (pp. 129 – 148). Cambridge, MA: MIT Press.

Müller, S., & Kasper, W. (2000). HPSG analysis of German. In W. Wahlster (Ed.), *Verbmobil. Foundations of speech-to-speech translation* (Artificial Intelligence ed., pp. 238 – 253). Berlin, Germany: Springer.

Oepen, S., & Carroll, J. (2002). Efficient parsing for unification-based grammars. In S. Oepen, D. Flickinger, J. Tsujii, & H. Uszkoreit (Eds.), *Collaborative language engineering. A case study in efficient grammar-based processing* (pp. 195 – 225). Stanford, CA: CSLI Publications.

Oepen, S., Flickinger, D., Toutanova, K., & Manning, C. D. (2004). LinGO Redwoods. A rich and dynamic treebank for HPSG. *Journal of Research on Language and Computation*, *2*(4), 575 – 596.

Riezler, S., King, T. H., Kaplan, R. M., Crouch, R., Maxwell III, J. T., & Johnson, M. (2002). Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Meeting of the Association for Computational Linguistics.* Philadelphia, PA.

Toutanova, K., Manning, C. D., Flickinger, D., & Oepen, S. (2005). Stochastic HPSG parse selection using the Redwoods corpus. *Journal of Research on Language and Computation*, *3*(1), 83 – 105.

A log-linear model with an n-gram reference distribution for accurate HPSG parsing

Takashi Ninomiya
Information Technology Center
University of Tokyo
ninomi@r.dl.itc.u-tokyo.ac.jp

Takuya Matsuzaki
Department of Computer Science
University of Tokyo
matuzaki@is.s.u-tokyo.ac.jp

Yusuke Miyao
Department of Computer Science
University of Tokyo
yusuke@is.s.u-tokyo.ac.jp

Jun'ichi Tsujii
Department of Computer Science, University of Tokyo
School of Informatics, University of Manchester
NaCTeM (National Center for Text Mining)
tsujii@is.s.u-tokyo.ac.jp
Hongo 7-3-1, Bunkyo-ku, Tokyo, 113-0033, Japan

Abstract

This paper describes a log-linear model with an n-gram reference distribution for accurate probabilistic HPSG parsing. In the model, the n-gram reference distribution is simply defined as the product of the probabilities of selecting lexical entries, which are provided by the discriminative method with machine learning features of word and POS n-gram as defined in the CCG/HPSG/CDG supertagging. Recently, supertagging becomes well known to drastically improve the parsing accuracy and speed, but supertagging techniques were heuristically introduced, and hence the probabilistic models for parse trees were not well defined. We introduce the supertagging probabilities as a reference distribution for the log-linear model of the probabilistic HPSG. This is the first model which properly incorporates the supertagging probabilities into parse tree's probabilistic model.

1 Introduction

For the last decade, fast, accurate and wide-coverage parsing for real-world text has been pursued in sophisticated grammar formalisms, such as head-driven phrase structure grammar (HPSG) (Pollard and Sag, 1994), combinatory categorial grammar (CCG) (Steedman, 2000) and lexical function grammar (LFG) (Bresnan, 1982). They are preferred because they give precise and in-depth analyses for explaining linguistic phenomena, such as passivization, control verbs and relative clauses. The main difficulty of developing parsers in these formalisms was how to model a well-defined probabilistic model for graph structures such as feature structures. This was overcome by a probabilistic model which provides probabilities of discriminating a correct parse tree among candidates of parse trees in a *log-linear model* or *maximum entropy model* (Berger et al., 1996) with many features for parse trees (Abney, 1997; Johnson et al., 1999; Riezler et al., 2000; Malouf and van Noord, 2004; Kaplan et al., 2004; Miyao and Tsujii, 2005). Following this discriminative approach, techniques for efficiency were investigated for estimation (Geman and Johnson, 2002; Miyao and Tsujii, 2002; Malouf and van Noord, 2004) and parsing (Clark and Curran, 2004b; Clark and Curran, 2004a; Ninomiya et al., 2005).

An interesting approach to the problem of parsing efficiency was using supertagging (Clark and Curran, 2004b; Clark and Curran, 2004a; Wang, 2003; Wang and Harper, 2004; Nasr and Rambow, 2004; Ninomiya et al., 2006; Foth et al., 2006; Foth and Menzel, 2006), which was originally developed for lexicalized tree adjoining grammars (LTAG) (Bangalore and Joshi, 1999). Supertagging is a process where words in an input sentence are tagged with 'supertags,' which are lexical entries in lexicalized grammars, e.g., elementary trees in LTAG, lexical categories in CCG, and lexical entries in HPSG. The concept of supertagging is simple and interesting, and the effects of this were recently demonstrated in the case of a CCG parser (Clark and Curran, 2004a) with the result of a drastic improvement in the parsing speed. Wang and Harper (2004) also demonstrated the effects of supertagging with a statistical constraint dependency grammar (CDG) parser by showing accuracy as high as the state-of-the-art parsers, and Foth et al. (2006) and Foth and Menzel (2006) reported that accuracy was significantly improved by incorporating the supertagging probabilities into manually tuned Weighted CDG. Ninomiya et al. (2006) showed the parsing model using only supertagging probabilities could achieve accuracy as high as the probabilistic model for phrase structures. This means that syntactic structures are almost determined by supertags as is claimed by Bangalore and Joshi (1999). However, supertaggers themselves were heuristically used as an external tagger. They filter out unlikely lexical entries just to help parsing (Clark and Curran, 2004a), or the probabilistic models for phrase structures were trained independently of the supertagger's probabilistic models (Wang and Harper, 2004; Ninomiya et al., 2006). In the case of supertagging of Weighted CDG (Foth et al., 2006), parameters for Weighted CDG are manually tuned, i.e., their model is not a well-defined probabilistic model.

We propose a log-linear model for probabilistic HPSG parsing in which the supertagging probabilities are introduced as a reference distribution for the probabilistic HPSG. The reference distribution is simply defined as the product of the probabilities of selecting lexical entries, which are provided by the discriminative method with machine learning features of word and part-of-speech (POS) n-gram as defined in the CCG/HPSG/CDG supertagging. This is the first model which properly incorporates the supertagging probabilities into parse tree's probabilistic model. We compared our model with the probabilistic model for phrase structures (Miyao and Tsujii, 2005). This model uses word and POS unigram for its reference distribution, i.e., the probabilities of unigram supertagging. Our model can be regarded as an extension of a unigram reference distribution to an n-gram reference distribution with features that are used in supertagging. We also compared with a probabilistic model in (Ninomiya et al., 2006). The probabilities of their model are defined as the product of probabilities of supertagging and probabilities of the probabilistic model for phrase structures, but their model was trained independently of supertagging probabilities, i.e., the supertagging probabilities are not used for reference distributions.

2 HPSG and probabilistic models

HPSG (Pollard and Sag, 1994) is a syntactic theory based on lexicalized grammar formalism. In HPSG, a small number of schemata describe general construction rules, and a large number of lexical entries express word-specific characteristics. The structures of sentences are explained using combinations of schemata and lexical entries. Both schemata and lexical entries are represented by typed feature structures, and constraints represented by feature structures are checked with *unification*.

An example of HPSG parsing of the sentence "*Spring has come*" is shown in Figure 1. First, each of the lexical entries for "*has*" and "*come*" is unified with a daughter feature structure of the Head-Complement Schema. Unification provides the phrasal sign of the mother. The sign of the larger constituent is obtained by repeatedly applying schemata to lexical/phrasal signs. Finally, the parse result is output as a phrasal sign that dominates the sentence.

Given a set $\mathcal{W}$ of words and a set $\mathcal{F}$ of feature structures, an HPSG is formulated as a tuple, $G = \langle L, R \rangle$, where

$L = \{l = \langle w, F \rangle | w \in \mathcal{W}, F \in \mathcal{F}\}$ is a set of lexical entries, and
R is a set of schemata; i.e., $r \in R$ is a partial function: $\mathcal{F} \times \mathcal{F} \to \mathcal{F}$.

Given a sentence, an HPSG computes a set of phrasal signs, i.e., feature structures, as a result of

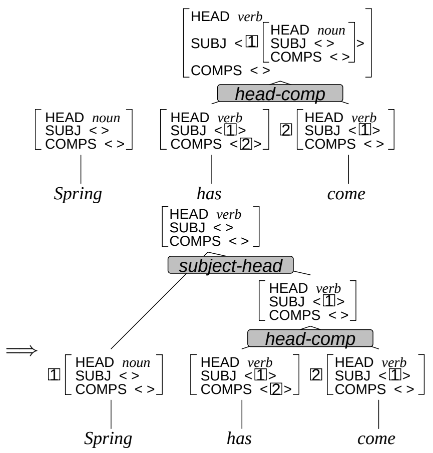


Figure 1: HPSG parsing.

parsing. Note that HPSG is one of the lexicalized grammar formalisms, in which lexical entries determine the dominant syntactic structures.

Previous studies (Abney, 1997; Johnson et al., 1999; Riezler et al., 2000; Malouf and van Noord, 2004; Kaplan et al., 2004; Miyao and Tsujii, 2005) defined a probabilistic model of unification-based grammars including HPSG as a *log-linear model* or *maximum entropy model* (Berger et al., 1996). The probability that a parse result T is assigned to a given sentence $\mathbf{w} = \langle w_1, \ldots, w_n \rangle$ is

(Probabilistic HPSG)

$$p_{hpsg}(T|\mathbf{w}) = \frac{1}{Z_{\mathbf{w}}} \exp\left(\sum_u \lambda_u f_u(T)\right)$$

$$Z_{\mathbf{w}} = \sum_{T'} \exp\left(\sum_u \lambda_u f_u(T')\right),$$

where λ_u is a model parameter, f_u is a feature function that represents a characteristic of parse tree T, and $Z_{\mathbf{w}}$ is the sum over the set of all possible parse trees for the sentence. Intuitively, the probability is defined as the normalized product of the weights $\exp(\lambda_u)$ when a characteristic corresponding to f_u appears in parse result T. The model parameters, λ_u, are estimated using numerical optimization methods (Malouf, 2002) to maximize the log-likelihood of the training data.

However, the above model cannot be easily estimated because the estimation requires the computation of $p(T|\mathbf{w})$ for all parse candidates assigned to sentence $\mathbf{w}$. Because the number of parse candidates is exponentially related to the length of the sentence, the estimation is intractable for long sentences. To make the model estimation tractable, Geman and Johnson (Geman and Johnson, 2002) and Miyao and Tsujii (Miyao and Tsujii, 2002) proposed a dynamic programming algorithm for estimating $p(T|\mathbf{w})$. Miyao and Tsujii (2005) also introduced a *preliminary probabilistic model* $p_0(T|\mathbf{w})$ whose estimation does not require the parsing of a treebank. This model is introduced as a *reference distribution* (Jelinek, 1998; Johnson and Riezler, 2000) of the probabilistic HPSG model; i.e., the computation of parse trees given low probabilities by the model is omitted in the estimation stage (Miyao and Tsujii, 2005), or a probabilistic model can be augmented by several distributions estimated from the larger and simpler corpus (Johnson and Riezler, 2000). In (Miyao and Tsujii, 2005), $p_0(T|\mathbf{w})$ is defined as the product of probabilities of selecting lexical entries with word and POS unigram features:

(Miyao and Tsujii (2005)'s model)

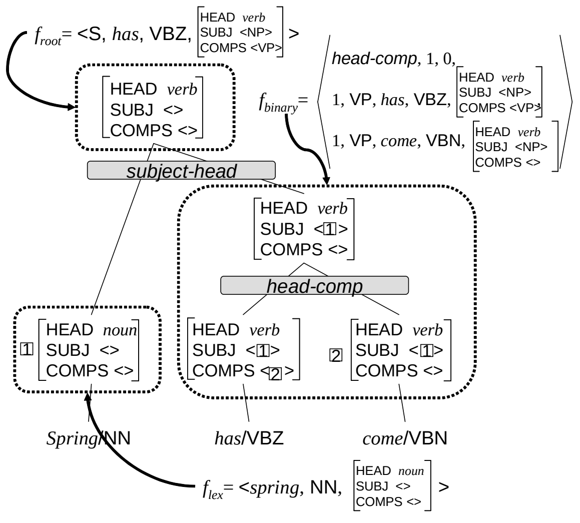
$$p_{uniref}(T|\mathbf{w}) = p_0(T|\mathbf{w}) \frac{1}{Z_{\mathbf{w}}} \exp\left(\sum_u \lambda_u f_u(T)\right)$$

$$Z_{\mathbf{w}} = \sum_{T'} p_0(T'|\mathbf{w}) \exp\left(\sum_u \lambda_u f_u(T')\right)$$

$$p_0(T|\mathbf{w}) = \prod_{i=1}^{n} p(l_i|w_i),$$

where l_i is a lexical entry assigned to word w_i in T and $p(l_i|w_i)$ is the probability of selecting lexical entry l_i for w_i.

In the experiments, we compared our model with other two types of probabilistic models using a supertagger (Ninomiya et al., 2006). The first one is the simplest probabilistic model, which is defined with only the probabilities of lexical entry selection. It is defined simply as the product of the probabilities of selecting all lexical entries in the sentence; i.e., the model does not use the probabilities of phrase structures like the probabilistic models explained above. Given a set of lexical entries, L, a sentence, $\mathbf{w} = \langle w_1, \ldots, w_n \rangle$, and the probabilistic model of lexical entry selection, $p(l_i \in L|\mathbf{w}, i)$, the first model is formally defined as follows:



f_{root}= <S, *has*, VBZ, [HEAD *verb* SUBJ <NP> COMPS <VP>] >
f_{binary}= < *head-comp*, 1, 0, 1, VP, *has*, VBZ, [HEAD *verb* SUBJ <NP> COMPS <VP>] 1, VP, *come*, VBN, [HEAD *verb* SUBJ <NP> COMPS <>] >
HEAD *verb* SUBJ <> COMPS <>
subject-head
HEAD *verb* SUBJ <1> COMPS <>
head-comp
1 HEAD *noun* SUBJ <> COMPS <>
HEAD *verb* SUBJ <1> COMPS <2>
2 HEAD *verb* SUBJ <1> COMPS <>
Spring/NN
has/VBZ
come/VBN
f_{lex}= <*spring*, NN, [HEAD *noun* SUBJ <> COMPS <>] >

Figure 2: Example of features.

(Ninomiya et al. (2006)'s model 1)

$$p_{model1}(T|\mathbf{w}) = \prod_{i=1}^{n} p(l_i|\mathbf{w}, i),$$

where l_i is a lexical entry assigned to word w_i in T and $p(l_i|\mathbf{w}, i)$ is the probability of selecting lexical entry l_i for w_i.

The probabilities of lexical entry selection, $p(l_i|\mathbf{w}, i)$, are defined as follows:

(Probabilistic model of lexical entry selection)

$$p(l_i|\mathbf{w}, i) = \frac{1}{Z_w} \exp\left(\sum_u \lambda_u f_u(l_i, \mathbf{w}, i)\right)$$

$$Z_w = \sum_{l'} \exp\left(\sum_u \lambda_u f_u(l', \mathbf{w}, i)\right),$$

where Z_w is the sum over all possible lexical entries for the word w_i.

The second model is a hybrid model of supertagging and the probabilistic HPSG. The probabilities are given as the product of Ninomiya et al. (2006)'s model 1 and the probabilistic HPSG.

(Ninomiya et al. (2006)'s model 3)

$$p_{model3}(T|\mathbf{w}) = p_{model1}(T|\mathbf{w}) p_{hpsg}(T|\mathbf{w})$$

In the experiments, we compared our model with Miyao and Tsujii (2005)'s model and Ninomiya et al. (2006)'s model 1 and 3. The features used in our model and their model are combinations of the feature templates listed in Table 1 and Table 2. The feature templates f_{binary} and f_{unary} are defined for constituents at binary and unary branches, f_{root} is a feature template set for the root nodes of parse trees. f_{lex} is a feature template set for calculating the unigram reference distribution and is used in Miyao and Tsujii (2005)'s model. f_{sptag} is a feature template set for calculating the probabilities of selecting lexical entries in Ninomiya et al. (2006)'s model 1 and 3. The feature templates in f_{sptag} are word trigrams and POS 5-grams. An example of features applied to the parse tree for the sentence "*Spring has come*" is shown in Figure 2.

$$\begin{aligned}
f_{binary} &= \left\langle \begin{array}{l} r, d, c, \\ sp_l, sy_l, hw_l, hp_l, hl_l, \\ sp_r, sy_r, hw_r, hp_r, hl_r \end{array} \right\rangle \\
f_{unary} &= \langle r, sy, hw, hp, hl \rangle \\
f_{root} &= \langle sy, hw, hp, hl \rangle \\
f_{lex} &= \langle w_i, p_i, l_i \rangle \\
f_{sptag} &= \left\langle \begin{array}{l} w_{i-1}, w_i, w_{i+1}, \\ p_{i-2}, p_{i-1}, p_i, p_{i+1}, p_{i+2} \end{array} \right\rangle
\end{aligned}$$

r	name of the applied schema
d	distance between the head words of the daughters
c	whether a comma exists between daughters and/or inside daughter phrases
sp	number of words dominated by the phrase
sy	symbol of the phrasal category
hw	surface form of the head word
hp	part-of-speech of the head word
hl	lexical entry assigned to the head word
w_i	i-th word
p_i	part-of-speech for w_i
l_i	lexical entry for w_i

Table 1: Feature templates.

3 Probabilistic HPSG with an n-gram reference distribution

In this section, we propose a probabilistic model with an n-gram reference distribution for probabilistic HPSG parsing. This is an extension of Miyao and Tsujii (2005)'s model by replacing the unigram reference distribution with an n-gram reference distribution. Our model is formally defined as follows:

combinations of feature templates for f_{binary}
$\langle r,d,c,hw,hp,hl\rangle$, $\langle r,d,c,hw,hp\rangle$, $\langle r,d,c,hw,hl\rangle$,
$\langle r,d,c,sy,hw\rangle$, $\langle r,c,sp,hw,hp,hl\rangle$, $\langle r,c,sp,hw,hp\rangle$,
$\langle r,c,sp,hw,hl\rangle$, $\langle r,c,sp,sy,hw\rangle$, $\langle r,d,c,hp,hl\rangle$,
$\langle r,d,c,hp\rangle$, $\langle r,d,c,hl\rangle$, $\langle r,d,c,sy\rangle$, $\langle r,c,sp,hp,hl\rangle$,
$\langle r,c,sp,hp\rangle$, $\langle r,c,sp,hl\rangle$, $\langle r,c,sp,sy\rangle$

combinations of feature templates for f_{unary}
$\langle r,hw,hp,hl\rangle$, $\langle r,hw,hp\rangle$, $\langle r,hw,hl\rangle$, $\langle r,sy,hw\rangle$,
$\langle r,hp,hl\rangle$, $\langle r,hp\rangle$, $\langle r,hl\rangle$, $\langle r,sy\rangle$

combinations of feature templates for f_{root}
$\langle hw,hp,hl\rangle$, $\langle hw,hp\rangle$, $\langle hw,hl\rangle$,
$\langle sy,hw\rangle$, $\langle hp,hl\rangle$, $\langle hp\rangle$, $\langle hl\rangle$, $\langle sy\rangle$

combinations of feature templates for f_{lex}
$\langle w_i,p_i,l_i\rangle$, $\langle p_i,l_i\rangle$

combinations of feature templates for f_{sptag}
$\langle w_{i-1}\rangle$, $\langle w_i\rangle$, $\langle w_{i+1}\rangle$,
$\langle p_{i-2}\rangle$, $\langle p_{i-1}\rangle$, $\langle p_i\rangle$, $\langle p_{i+1}\rangle$, $\langle p_{i+2}\rangle$, $\langle p_{i+3}\rangle$,
$\langle w_{i-1},w_i\rangle$, $\langle w_i,w_{i+1}\rangle$,
$\langle p_{i-1},w_i\rangle$, $\langle p_i,w_i\rangle$, $\langle p_{i+1},w_i\rangle$,
$\langle p_i,p_{i+1},p_{i+2},p_{i+3}\rangle$, $\langle p_{i-2},p_{i-1},p_i\rangle$,
$\langle p_{i-1},p_i,p_{i+1}\rangle$, $\langle p_i,p_{i+1},p_{i+2}\rangle$
$\langle p_{i-2},p_{i-1}\rangle$, $\langle p_{i-1},p_i\rangle$, $\langle p_i,p_{i+1}\rangle$, $\langle p_{i+1},p_{i+2}\rangle$

Table 2: Combinations of feature templates.

(Probabilistic HPSG with an n-gram reference distribution)

$$
\begin{aligned}
&p_{nref}(T|\mathbf{w}) = \\
&\quad \frac{1}{Z_{nref}} p_{model1}(T|\mathbf{w}) \exp\left(\sum_u \lambda_u f_u(T)\right) \\
&Z_{nref} = \\
&\quad \sum_{T'} p_{model1}(T'|\mathbf{w}) \exp\left(\sum_u \lambda_u f_u(T')\right).
\end{aligned}
$$

In our model, Ninomiya et al. (2006)'s model 1 is used as a reference distribution. The probabilistic model of lexical entry selection and its feature templates are the same as defined in Ninomiya et al. (2006)'s model 1.

The formula of our model is the same as Ninomiya et al. (2006)'s model 3. But, their model is not a probabilistic model with a reference distribution. Both our model and their model consist of the probabilities for lexical entries (= $p_{model1}(T|\mathbf{w})$) and the probabilities for phrase structures (= the rest of each formula). The only difference between our model and their model is the way of how to train model parameters for phrase structures. In both our model and their model, the parameters for lexical entries (= the parameters of $p_{model1}(T|\mathbf{w})$) are first estimated from the word and POS sequences independently of the parameters for phrase structures. That is, the estimated parameters for lexical entries are the same in both models, and hence the probabilities of $p_{model1}(T|\mathbf{w})$ of both models are the same. Note that the parameters for lexical entries will never be updated after this estimation stage; i.e., the parameters for lexical entries are not estimated in the same time with the parameters for phrase structures. The difference of our model and their model is the estimation of parameters for phrase structures. In our model, given the probabilities for lexical entries, the parameters for phrase structures are estimated so as to maximize the entire probabilistic model (= the product of the probabilities for lexical entries and the probabilities for phrase structures) in the training corpus. In their model, the parameters for phrase structures are trained without using the probabilities for lexical entries, i.e., the parameters for phrase structures are estimated so as to maximize the probabilities for phrase structures only. That is, the parameters for lexical entries and the parameters for phrase structures are trained independently in their model.

Miyao and Tsujii (2005)'s model also uses a reference distribution, but with word and POS unigram features, as is explained in the previous section. The only difference between our model and Miyao and Tsujii (2005)'s model is that our model uses sequences of word and POS tags as n-gram features for selecting lexical entries in the same way as supertagging does.

4 Experiments

We evaluated the speed and accuracy of parsing by using Enju 2.1, the HPSG grammar for English (Miyao et al., 2005; Miyao and Tsujii, 2005). The lexicon of the grammar was extracted from Sections 02-21 of the Penn Treebank (Marcus et al., 1994) (39,832 sentences). The grammar consisted of 3,797 lexical entries for 10,536 words[1]. The prob-

[1]An HPSG treebank is automatically generated from the Penn Treebank. Those lexical entries were generated by applying lexical rules to observed lexical entries in the HPSG treebank (Nakanishi et al., 2004). The lexicon, however, included many lexical entries that do not appear in the HPSG treebank.

	No. of tested sentences	Total No. of sentences	Avg. length of tested sentences
Section 23	2,299 (100.00%)	2,299	22.2
Section 24	1,245 (99.84%)	1,247	23.0

Table 3: Statistics of the Penn Treebank.

	Section 23 (Gold POSs)						
	LP (%)	LR (%)	LF (%)	UP (%)	UR (%)	UF (%)	Avg. time (ms)
Miyao and Tsujii (2005)'s model	87.26	86.50	86.88	90.73	89.93	90.33	604
Ninomiya et al. (2006)'s model 1	87.23	86.47	86.85	90.05	89.27	89.66	129
Ninomiya et al. (2006)'s model 3	89.48	88.58	89.02	92.33	91.40	91.86	152
our model 1	89.78	89.28	89.53	92.58	92.07	92.32	234
our model 2	90.03	89.60	89.82	92.82	92.37	92.60	1379
	Section 23 (POS tagger)						
	LP (%)	LR (%)	LF (%)	UP (%)	UR (%)	UF (%)	Avg. time (ms)
Miyao and Tsujii (2005)'s model	84.96	84.25	84.60	89.55	88.80	89.17	674
Ninomiya et al. (2006)'s model 1	85.00	84.01	84.50	88.85	87.82	88.33	154
Ninomiya et al. (2006)'s model 3	87.35	86.29	86.82	91.24	90.13	90.68	183
Matsuzaki et al. (2007)'s model	86.93	86.47	86.70	-	-	-	30
our model 1	87.28	87.05	87.17	91.62	91.38	91.50	260
our model 2	87.56	87.46	87.51	91.88	91.77	91.82	1821

Table 4: Experimental results for Section 23.

abilistic models were trained using the same portion of the treebank. We used beam thresholding, global thresholding (Goodman, 1997), preserved iterative parsing (Ninomiya et al., 2005) and quick check (Malouf et al., 2000).

We measured the accuracy of the predicate-argument relations output of the parser. A predicate-argument relation is defined as a tuple $\langle \sigma, w_h, a, w_a \rangle$, where σ is the predicate type (e.g., adjective, intransitive verb), w_h is the head word of the predicate, a is the argument label (MODARG, ARG1, ..., ARG4), and w_a is the head word of the argument. Labeled precision (LP)/labeled recall (LR) is the ratio of tuples correctly identified by the parser[2]. Unlabeled precision (UP)/unlabeled recall (UR) is the ratio of tuples without the predicate type and the argument label. This evaluation scheme was the same as used in previous evaluations of lexicalized grammars (Hockenmaier, 2003; Clark and Curran, 2004b; Miyao and Tsujii, 2005). The experiments were conducted on an AMD Opteron server with a 2.4-GHz CPU. Section 22 of the Treebank was used as the development set, and the performance was evaluated using sentences of $\leq$ 100 words in Section 23. The performance of each model was analyzed using the sentences in Section 24 of $\leq$ 100 words. Table 3 details the numbers and average lengths of the tested sentences of $\leq$ 100 words in Sections 23 and 24, and the total numbers of sentences in Sections 23 and 24.

The parsing performance for Section 23 is shown in Table 4. The upper half of the table shows the performance using the correct POSs in the Penn Treebank, and the lower half shows the performance using the POSs given by a POS tagger (Tsuruoka and Tsujii, 2005). LF and UF in the figure are labeled F-score and unlabeled F-score. F-score is the harmonic mean of precision and recall. We evaluated our model in two settings. One is implemented with a narrow beam width ('our model 1' in the figure), and the other is implemented with a wider beam width ('our model 2' in the figure)[3]. 'our model

The HPSG treebank is used for training the probabilistic model for lexical entry selection, and hence, those lexical entries that do not appear in the treebank are rarely selected by the probabilistic model. The 'effective' tag set size, therefore, is around 1,361, the number of lexical entries without those never-seen lexical entries.

[2]When parsing fails, precision and recall are evaluated, although nothing is output by the parser; i.e., recall decreases greatly.

[3]The beam thresholding parameters for 'our model 1' are $\alpha_0 = 10, \Delta\alpha = 5, \alpha_{\text{last}} = 30, \beta_0 = 5.0, \Delta\beta = 2.5, \beta_{\text{last}} = 15.0, \delta_0 = 10, \Delta\delta = 5, \delta_{\text{last}} = 30$, $\kappa_0 = 5.0, \Delta\kappa = 2.5, \kappa_{\text{last}} = 15.0$, $\theta_0 = 6.0, \Delta\theta = 3.5$, and $\theta_{\text{last}} = 20.0$.

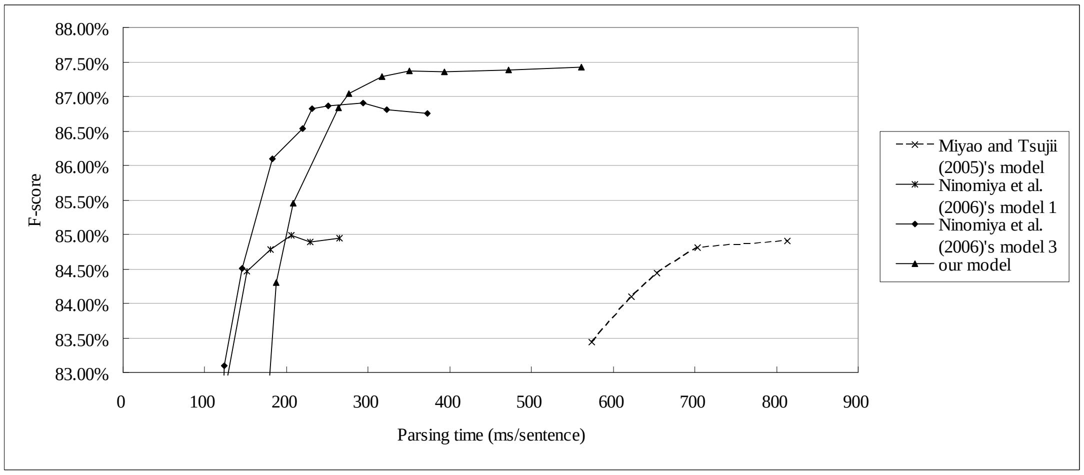


Figure 3: F-score versus average parsing time for sentences in Section 24 of $\leq$ 100 words.

1' was introduced to measure the performance with balanced F-score and speed, which we think appropriate for practical use. 'our model 2' was introduced to measure how high the precision and recall could reach by sacrificing speed. Our models increased the parsing accuracy. 'our model 1' was around 2.6 times faster and had around 2.65 points higher F-score than Miyao and Tsujii (2005)'s model. 'our model 2' was around 2.3 times slower but had around 2.9 points higher F-score than Miyao and Tsujii (2005)'s model. We must admit that the difference between our models and Ninomiya et al. (2006)'s model 3 was not as great as the difference from Miyao and Tsujii (2005)'s model, but 'our model 1' achieved 0.56 points higher F-score, and 'our model 2' achieved 0.8 points higher F-score. When the automatic POS tagger was introduced, F-score dropped by around 2.4 points for all models.

We also compared our model with Matsuzaki et al. (2007)'s model. Matsuzaki et al. (2007) proposed a technique for efficient HPSG parsing with supertagging and CFG filtering. Their results with the same grammar and servers are also listed in the lower half of Table 4. They achieved drastic improvement in efficiency. Their parser ran around 6 times faster than Ninomiya et al. (2006)'s model 3, 9 times faster than 'our model 1' and 60 times faster than 'our model 2.' Instead, our models achieved better accuracy. 'our model 1' had around 0.5 higher F-score, and 'our model 2' had around 0.8 points higher F-score. Their efficiency is mainly due to elimination of ungrammatical lexical entries by the CFG filtering. They first parse a sentence with a CFG grammar compiled from an HPSG grammar, and then eliminate lexical entries that are not in the parsed CFG trees. Obviously, this technique can also be applied to the HPSG parsing of our models. We think that efficiency of HPSG parsing with our models will be drastically improved by applying this technique.

The average parsing time and labeled F-score curves of each probabilistic model for the sentences in Section 24 of $\leq$ 100 words are graphed in Figure 3. The graph clearly shows the difference of our model and other models. As seen in the graph, our model achieved higher F-score than other model when beam threshold was widen. This implies that other models were probably difficult to reach the F-score of 'our model 1' and 'our model 2' for Section 23 even if we changed the beam thresholding parameters. However, F-score of our model dropped eas-

The terms κ and δ are the thresholds of the number of phrasal signs in the chart cell and the beam width for signs in the chart cell. The terms α and β are the thresholds of the number and the beam width of lexical entries, and θ is the beam width for global thresholding (Goodman, 1997). The terms with suffixes 0 are the initial values. The parser iterates parsing until it succeeds to generate a parse tree. The parameters increase for each iteration by the terms prefixed by Δ, and parsing finishes when the parameters reach the terms with suffixes last. Details of the parameters are written in (Ninomiya et al., 2005). The beam thresholding parameters for 'our model 2' are $\alpha_0 = 18, \Delta\alpha = 6, \alpha_{\text{last}} = 42, \beta_0 = 9.0, \Delta\beta = 3.0, \beta_{\text{last}} = 21.0, \delta_0 = 18, \Delta\delta = 6, \delta_{\text{last}} = 42$, $\kappa_0 = 9.0, \Delta\kappa = 3.0, \kappa_{\text{last}} = 21.0$. In 'our model 2', the global thresholding was not used.

ily when we narrow down the beam threshold, compared to other models. We think that this is mainly due to its bad implementation of parser interface. The n-gram reference distribution is incorporated into the kernel of the parser, but the n-gram features and a maximum entropy estimator are defined in other modules; n-gram features are defined in a grammar module, and a maximum entropy estimator for the n-gram reference distribution is implemented with a general-purpose maximum entropy estimator module. Consequently, strings that represent the n-gram information are very frequently changed into feature structures and vice versa when they go in and out of the kernel of the parser. On the other hand, Ninomiya et al. (2006)'s model 3 uses the supertagger as an external module. Once the parser acquires the supertagger's outputs, the n-gram information never goes in and out of the kernel. This advantage of Ninomiya et al. (2006)'s model can apparently be implemented in our model, but this requires many parts of rewriting of the implemented parser. We estimate that the overhead of the interface is around from 50 to 80 ms/sentence. We think that re-implementation of the parser will improve the parsing speed as estimated. In Figure 3, the line of our model crosses the line of Ninomiya et al. (2006)'s model. If the estimation is correct, our model will be faster and more accurate so that the lines in the figure do not cross. Speed-up in our model is left as a future work.

5 Conclusion

We proposed a probabilistic model in which supertagging is consistently integrated into the probabilistic model for HPSG. In the model, the n-gram reference distribution is simply defined as the product of the probabilities of selecting lexical entries with machine learning features of word and POS n-gram as defined in the CCG/HPSG/CDG supertagging. We conducted experiments on the Penn Treebank with a wide-coverage HPSG parser. In the experiments, we compared our model with the probabilistic HPSG with a unigram reference distribution (Miyao and Tsujii, 2005) and the probabilistic HPSG with supertagging (Ninomiya et al., 2006). Though our model was not as fast as Ninomiya et al. (2006)'s models, it achieved the highest accuracy among them. Our model had around 2.65 points higher F-score than Miyao and Tsujii (2005)'s model and around 0.56 points higher F-score than the Ninomiya et al. (2006)'s model 3. When we sacrifice parsing speed, our model achieved around 2.9 points higher F-score than Miyao and Tsujii (2005)'s model and around 0.8 points higher F-score than Ninomiya et al. (2006)'s model 3. Our model achieved higher F-score because parameters for phrase structures in our model are trained with the supertagging probabilities, which are not in other models.

References

Steven P. Abney. 1997. Stochastic attribute-value grammars. *Computational Linguistics*, 23(4):597–618.

Srinivas Bangalore and Aravind Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

Adam Berger, Stephen Della Pietra, and Vincent Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Joan Bresnan. 1982. *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, MA.

Stephen Clark and James R. Curran. 2004a. The importance of supertagging for wide-coverage CCG parsing. In *Proc. of COLING-04*.

Stephen Clark and James R. Curran. 2004b. Parsing the WSJ using CCG and log-linear models. In *Proc. of ACL'04*, pages 104–111.

Killian Foth and Wolfgang Menzel. 2006. Hybrid parsing: Using probabilistic models as predictors for a symbolic parser. In *Proc. of COLING-ACL 2006*.

Killian Foth, Tomas By, and Wolfgang Menzel. 2006. Guiding a constraint dependency parser with supertags. In *Proc. of COLING-ACL 2006*.

Stuart Geman and Mark Johnson. 2002. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proc. of ACL'02*, pages 279–286.

Joshua Goodman. 1997. Global thresholding and multiple pass parsing. In *Proc. of EMNLP-1997*, pages 11–25.

Julia Hockenmaier. 2003. Parsing with generative models of predicate-argument structure. In *Proc. of ACL'03*, pages 359–366.

F. Jelinek. 1998. *Statistical Methods for Speech Recognition*. The MIT Press.

Mark Johnson and Stefan Riezler. 2000. Exploiting auxiliary distributions in stochastic unification-based grammars. In *Proc. of NAACL-2000*, pages 154–161.

Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic “unification-based” grammars. In *Proc. of ACL ’99*, pages 535–541.

R. M. Kaplan, S. Riezler, T. H. King, J. T. Maxwell III, and A. Vasserman. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proc. of HLT/NAACL’04*.

Robert Malouf and Gertjan van Noord. 2004. Wide coverage parsing with stochastic attribute value grammars. In *Proc. of IJCNLP-04 Workshop “Beyond Shallow Analyses”*.

Robert Malouf, John Carroll, and Ann Copestake. 2000. Efficient feature structure operations without compilation. *Journal of Natural Language Engineering*, 6(1):29–46.

Robert Malouf. 2002. A comparison of algorithms for maximum entropy parameter estimation. In *Proc. of CoNLL-2002*, pages 49–55.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Takuya Matsuzaki, Yusuke Miyao, and Jun’ichi Tsujii. 2007. Efficient HPSG parsing with supertagging and CFG-filtering. In *Proc. of IJCAI 2007*, pages 1671–1676.

Yusuke Miyao and Jun’ichi Tsujii. 2002. Maximum entropy estimation for feature forests. In *Proc. of HLT 2002*, pages 292–297.

Yusuke Miyao and Jun’ichi Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proc. of ACL’05*, pages 83–90.

Yusuke Miyao, Takashi Ninomiya, and Jun’ichi Tsujii, 2005. *Keh-Yih Su, Jun’ichi Tsujii, Jong-Hyeok Lee and Oi Yee Kwong (Eds.), Natural Language Processing - IJCNLP 2004 LNAI 3248*, chapter Corpus-oriented Grammar Development for Acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank, pages 684–693. Springer-Verlag.

Hiroko Nakanishi, Yusuke Miyao, and Jun’ichi Tsujii. 2004. An empirical investigation of the effect of lexical rules on parsing with a treebank grammar. In *Proc. of TLT’04*, pages 103–114.

Alexis Nasr and Owen Rambow. 2004. Supertagging and full parsing. In *Proc. of the 7th International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+7)*.

Takashi Ninomiya, Yoshimasa Tsuruoka, Yusuke Miyao, and Jun’ichi Tsujii. 2005. Efficacy of beam thresholding, unification filtering and hybrid parsing in probabilistic HPSG parsing. In *Proc. of IWPT 2005*, pages 103–114.

Takashi Ninomiya, Takuya Matsuzaki, Yoshimasa Tsuruoka, Yusuke Miyao, and Jun’ichi Tsujii. 2006. Extremely lexicalized models for accurate and fast HPSG parsing. In *Proc. of EMNLP 2006*, pages 155–163.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

Stefan Riezler, Detlef Prescher, Jonas Kuhn, and Mark Johnson. 2000. Lexicalized stochastic modeling of constraint-based grammars using log-linear measures and EM training. In *Proc. of ACL’00*, pages 480–487.

Mark Steedman. 2000. *The Syntactic Process*. The MIT Press.

Yoshimasa Tsuruoka and Jun’ichi Tsujii. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proc. of HLT/EMNLP 2005*, pages 467–474.

Wen Wang and Mary P. Harper. 2004. A statistical constraint dependency grammar (CDG) parser. In *Proc. of ACL’04 Incremental Parsing workshop: Bringing Engineering and Cognition Together*, pages 42–49.

Wen Wang. 2003. *Statistical Parsing and Language Modeling based on Constraint Dependency Grammar*. Ph.D. thesis, Purdue University.

Ambiguity Resolution by Reordering Rules in Text Containing Errors

Sylvana Sofkova Hashemi
Department of Linguistics, Göteborg University
Box 200, SE-405 30 Göteborg, SWEDEN
sylvana@ling.gu.se

Abstract

Writing aids such as spelling and grammar checkers are often based on texts by adult writers and are not sufficiently targeted to support children in their writing process. This paper reports on the development of a writing tool based on a corpus of Swedish text written by children and on the parsing methods developed to handle text containing errors. The system uses finite state techniques for finding grammar errors without actually specifying the error. The 'broadness' of the grammar and the lexical ambiguity in words, necessary for parsing text containing errors, also yields ambiguous and/or alternative phrase annotations. We block some of the (erroneous) alternative parses by the order in which phrase segments are selected, which causes bleeding of some rules and more 'correct' parsing results are achieved. The technique shows good coverage results for agreement and verb selection phenomena.

1 Introduction

Writing on a computer in school often involves making a fair copy from a handwritten draft. Although a computer is an excellent means for the writing process, especially the linguistic tools are not used adequately. Spelling and grammar correctors are in general developed for and adapted to adult writers and have difficulties to support children in their writing development and give no space for acquisition or training. Errors in texts written by school children are more frequent and the distribution of the error types is different from adult writers.

This paper reports on the development of a finite state system for finding grammar errors, called *FiniteCheck*, based on a corpus of Swedish text written by school children. The system applies descriptions of correct language use in the detection process of grammatical violations and contains no rules describing the nature of the erroneous segments the system searches for. The approach (following Karttunen et al., 1996) for finding errors involves developing automata that represent two 'positive' grammars with varying degree of detail and then subtracting the detailed one from the general one. The difference between the automata corresponds to a grammar for errors.

2 Grammar Checkers

2.1 Current Systems

Whereas *spelling checkers* are standard in most word processors, grammar checking is a rather recent technology, especially for Swedish. Different methods and techniques have been applied to handle *nonsense words* and thus operate on isolated words as most spelling correctors do. Both statistical and rule-based methods and also algorithms that to some extent take into consideration the surrounding context (i.e. context-sensitive errors) or how a word is pronounced have been used for spelling correction (cf. Kukich, 1992).

Grammar checkers involve techniques and solve problems above the single word level and require syntactic, semantic or even discourse analysis (see Section 2.2). Grammar checking techniques started to develop first in the 1980's with products mainly for English (see Jensen et al, 1993; Vernon, 2000) but also for other languages, e.g. French (Chanod, 1996), Dutch (Vosse, 1994), Czech (Kirschner,

Proceedings of the 10th Conference on Parsing Technologies, pages 69–79,
Prague, Czech Republic, June 2007. ©2007 Association for Computational Linguistics

1994), Spanish and Greek (Bustamente and León, 1996). Computer-based grammar checking for Swedish is fairly recent and has primarily focused on the needs of adult writers. The first product release of such a writing aid was in November 1998 with the tool *Grammatifix* (Arppe, 2000; Birn, 2000), now part of the Swedish Microsoft Office 2000. Two other research groups developed grammar checking prototypes: *Granska* (Knutsson, 2001; Domeij, 2003) and *Scarrie* (Sågvall Hein, 1999).

2.2 Methods and Techniques

Many of the grammar checking systems are commercial products and technical documentation is often minimal or even absent. *Critique* (known until 1984 as *Epistle*) is an exception, a system developed in collaboration with IBM within the *Programming Language for Natural Language Processing* (PLNLP) project (Jensen et al., 1993). This tool is based on a parser using *Augmented Phrase Structure Grammar* (ACFG) and produces a complete analysis for all sentences (even ungrammatical) by application of *relaxation* rules when parsing fails on the first try or *parse fitting* procedure identifying the head and its constituents (Heidorn, 1993; Jensen et al., 1993). This approach of providing analysis of all sentences had influenced other grammar formalisms such as *Constraint Grammar* (Karlsson et al., 1995) or *Functional Dependency Grammar* (Järvinen and Tapanainen, 1998). The methods of *rule relaxation* and *parse fitting* had an impact on the development of other grammar checking systems.

The three Swedish tools use different technology to analyze unrestricted text and detect grammar errors. The lexical analysis in *Grammatifix* is based on the morphological analyzer *SWETWOL*, designed according to the principles of *two-level morphology* (Karlsson, 1992). The part-of-speech assignment applies the *Swedish Constraint Grammar* (SWECG), a surface-syntactic parser applying context-sensitive disambiguation rules (Birn, 1998). Errors are detected by *partial parsing* and *relaxation* on rules, regarding certain word sequences as phrases despite grammar errors in them.

Granska combines *probabilistic* and *rule-based* methods, where specific error rules (around 600) and local applied rules detect ungrammaticalities in free text. The lexical analyzer applies *Hidden Markov Models* and a rule matching system analyses the tagged text searching for grammatical violations defined in the detection rules and produces error description and a correction suggestion for the error (Carlberger & Kann, 1999).

The grammar checker in *Scarrie* is based on a previously developed parser, the *Uppsala Chart Parser* (UCP), a procedural, bottom-up parser, applying a longest path strategy (Sågvall Hein, 1983). The parsing strategy of erroneous input is based on *constraint relaxation* and application of *local error rules*. The grammar is in other words underspecified to a certain level, allowing feature violations and parsing of ungrammatical word sequences (Wedbjer Rambell, 1999).

The Swedish approaches to detection of grammar errors vary from chart-based methods in *Scarrie*, application of constraint grammars in *Grammatifix*, to a combination of probabilistic and rule-based methods in *Granska*. *Scarrie* and *Granska* identify erroneous patterns by partial analysis, whereas *Grammatifix* produces full analysis for both grammatical and ungrammatical sentences. All the tools define (wholly or to some extent) explicit error rules describing the nature of the error they search for. In the process of error detection they either proceed sentence by sentence, requiring recognition of sentence boundaries, or they rely in their rules on for instance capitalization conventions.

2.3 Error Coverage

Current grammar checking systems are restricted to a small set of all possible writing errors, concerning mostly syntactic analysis. The choice of what types of errors are detected in the Swedish tools is based on analysis of errors in writing of certain groups of writers (e.g. professional writers, writers at work). The coverage of error types is very similar between the systems, including errors in noun phrase agreement and agreement in predicative complement, pronoun case after preposition, word order, errors in verbs, etc.

Observations with children writing on a computer in school (Hård af Segerstad & Sofkova Hashemi, 2006; Sofkova Hashemi, forthcoming) and performance tests of the Swedish tools on texts written by school children (Sofkova Hashemi, 2003) show that grammar checkers do not sufficiently support school children in their writing development. The grammatical mistakes found in texts written by children display different frequency and distribution than in adults and the text structure as whole is different. Main clauses are often joined together without conjunctions and punctuation marks often delimit larger textual units than syntactic sentences. Sentence boundaries and capitalization are something the Swedish tools rely on in their detection process, which may have impact on the coverage results. Although the systems cover many of the types of errors found in school texts, they detect around 12% of all writing errors (Sofkova Hashemi, 2003) (see Section 6). Performance on text data such as newspaper texts and student compositions evaluated within the frames of the separate projects shows a much higher coverage of error detection on average 58% (Birn, 2000; Knutsson, 2001; Sågvall Hein et al., 1999).

3 The Training Data

3.1 The Child Data Corpus

FiniteCheck, the grammar error detector reported in this paper, is based on a corpus of Swedish text written by school children. This *Child Data* corpus of 29 812 words (3 373 word types) is composed of computer written and hand written essays written by children between 9 and 13 years of age. In general, the text structure of the compositions reveals clearly the influence of spoken language and performance difficulties in spelling, segmentation of words, the use of capitals and punctuation, with fairly wide variation both by individual and age. In total, 260 instances of grammatical errors were found in 134 narratives.

3.2 The Error Types

The most frequent grammatical violation concerns *the omission of finite verb inflection* (42% of all errors), i.e. when the main finite verb in a clause lacks the appropriate present or past tense endings:

(1) *På natten* ****vakna*** *jag av att brandlarmet tjöt*
in the-night wake[untensed] I from that fire-alarm howled
– In the night I woke up from that the fire-alarm went off.

The correct form of the verb *vakna* 'wake' should be in the past tense, i.e. *vaknade* 'woke'. This type of error arises from the fact that the writing is highly influenced by spoken language. In spoken Swedish regular weak verbs in the past tense often lack the appropriate ending and the spoken form then coincides with the infinitive (and for some verbs also imperative) form of the verb.

Other frequent grammar problems concern *extra inserted or missing words* in sentences (22%), here the preposition *i* 'in' is missing:

(2) *Gunnar var på semester* ****_ norge*** *och åkte skidor.*
Gunnar was on vacation _ Norway and went skis
– Gunnar was on vacation in Norway and skied.

word choice errors (11%), here the verb *att vara lika* 'to be alike' requires the particle *till* 'to' in combination with the noun phrase *sättet* 'the-manner' and not *på* 'on' as the writer uses:

(3) *vi var väldigt* ***lika *på sättet***
we were very like on the-manner
– We were very alike in the manners.

errors in *noun phrase agreement* (6%), here the correct form of the noun phrase requires the noun to be definite as in *den närmsta handduken* 'the nearest towel':

(4) *jag tar* ***den närmsta *handduk*** *och slänger den i vasken*
I take the[def] nearest[def] towel [indef] and throw it in the sink
– I take the nearest towel and throw it into the sink.

errors in *verb chains* (3%), here the auxiliary verb should be followed by an infinitive, *ska bli* 'will become', but in this case the present tense is used:

(5) *Men kom ihåg att det inte* ***ska *blir*** *någon riktig brand.*
but remember that it not will becomes[pres] some real fire
– But remember that there will not be a real fire.

Other grammar errors occurred less than ten times in the whole corpus, including reference errors, agreement between subject and predicative com-

plement, definiteness in single nouns, pronoun form, errors in infinitive phrases, word order.
Punctuation problems are also included in the analyses. In general, the use of punctuation varies from no usage at all (mostly among the youngest children) to rather sparse marking. In the following example the main clauses are joined together and the boundary between the sentences is not marked:

(6) *nasse blev arg han gick och la sig med dom andra syskonen.*
nasse became angry he went and lay himself with the other siblings
– Nasse got angry. He went and lay down with the other siblings.

The finite verb problem, verb form in verb chains and infinitive phrases and agreement problems in noun phrase are the four types of errors detected by the current system, *FiniteCheck*.

4 System architecture

The framework for detection of grammar errors in *FiniteCheck* is built as a network of finite state transducers compiled from regular expressions including operators defined in the *Xerox Finite State Tool* (XFST) (Karttunen et al., 1997). Each automaton in the network composes with the result of previous application and in principle all the automata can be composed into a single transducer.

There are in general two types of transducers in use: one that annotates text in order to select certain segments and one that redefines or refines earlier decisions. Annotations of any kind are handled by transducers defined as *finite state markers* that add reserved symbols into text and mark out syntactical segments, grammar errors, or other patterns aimed at selections. *Finite state filters* are used for refinement and/or revision of earlier decisions.

The system runs under UNIX in a simple *Emacs* environment used for testing and development of finite state grammars. The environment shows the results of an XFST-process run on the current Emacs buffer in a separate buffer. An XFST-mode allows for menus to be used and recompile files in the system.

The sequenced finite state transducers of *FiniteCheck* are divided in four main modules: *the lexicon lookup, the grammar, the parser* and *the error finder* – see Figure 1.

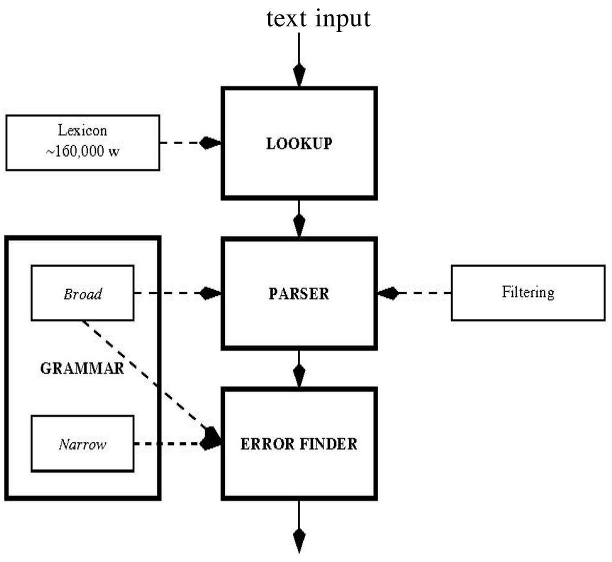


Figure 1: The system architecture

4.1 The Lexicon Lookup

The lexicon of around 160, 000 word forms, is built as a finite state transducer, using the Xerox tool *Finite State Lexicon Compiler* (Karttunen, 1993). The lexicon is composed from two resources and takes a string and maps inflected surface form to a tag containing part-of-speech and feature information, e. g. applying the transducer to the string *kvinna* 'woman' will return [nn utr sin ind nom]. The morphosyntactic tags follow directly the relevant string or token. More than one tag can be attached to a string, since no contextual information is taken into account. The morphosyntactic information in the tags is further used in the grammars of the system. The set of tags follows the *Stockholm-Umeå Corpus* project conventions (Ejerhed et al, 1992), including 23 category classes and 29 feature classes that were extended with 3 additional categories. Below is an example of a lookup on the example sentence in (5):

(7) Men[kn] kom[qmvb prt akt][vb prt akt] ihåg[ab][pl] att[sn][ie] det[pn neu sin def sub/obj] [dt neu sin def] inte[ab] ska[vb prs akt][mvb prs akt] blir[vb prs akt] någon[dt utr sin ind][pn utr sin ind sub/obj] riktig[jj pos utr sin ind nom] brand[nn utr sin ind nom]

4.2 The Grammar

The grammar module includes two grammar sets with (positive) rules reflecting the grammatical structure of Swedish, differing in the level of detail. The *broad grammar* is especially designed to handle text with ungrammaticalities and the linguistic descriptions are less accurate accepting both valid and invalid patterns. The *narrow gramar* is fine and accurate and accepts only the grammatical segments. For example, the regular expression in (8) belongs to the broad grammar set and recognizes potential *verb clusters* (VC) (both grammatical and ungrammatical) as a pattern consisting of a sequence of two or three verbs in combination with (zero or more) adverbs:

(8) define VC [Verb Adv* Verb (Verb)];

This automaton accepts all the verb cluster examples in (9), including the ungrammatical instance (9c) (marked by an asterisk '*'), where a finite verb follows a (finite) auxiliary verb.

(9) a. *kan inte springa* 'can not run'
    b. *skulle ha sprungit* 'would have run [sup]'
    c. **ska blir* 'will be [pres]'

Corresponding rules in the narrow grammar set represented by the regular expressions in (10) take into account the internal structure of a verb cluster and define the grammar of modal auxiliary verbs (Mod) followed by (zero or more) adverb(s), and either a verb in infinitive form (VerbInf) as in (10a), or a temporal verb in infinitive (PerfInf) and a verb in supine form (VerbSup), as in (10b). These rules thus accept only the grammatical segments in (9) and will not include example (9c). The actual grammar of grammatical verb clusters is a little bit more complex.

(10) a. define VC1 [Mod Adv* VerbInf];
     b. define VC2 [Mod Adv* PerfInf VerbSup];

4.3 The parser

The various kinds of constituents are marked out in a text using a *lexical-prefix-first* method, i.e. parsing first from left margin of a phrase to the head and then extending the phrase by adding on complements. The actual parsing (based on the *broad grammar* definitions) is incremental in a similar fashion as the methods described in Ait-Mohtar and Chanod (1997), where the output from one layer serves as input to the next, building on the segments. The system recognizes the head phrases in certain order in the first phase (verbal head, prepositional head, adjective phrase) and then applies the second phase in the reverse order and extends the phrases with complements (noun phrase, prepositional phrase, verb phrase). The parsing method is described in detail in Section 5.

4.4 Error Detection

The error finder is a separate module in the system, which means that the grammar and parser could potentially be used directly in a different application. The nets of this module correspond to the difference between the two grammars, *broad* and *narrow*.

By subtracting the narrow grammar from the broad grammar we create machines that will find ungrammatical phrases in a text. For example, the regular expression in (11) identifies verb clusters that violate the narrow grammar of modal verb clusters (VC1 or VC2, defined in (10)) by subtracting these rules from the more general (overgenerating) rule in the broad grammar (VC, defined in (8)) within the boundaries of a verb cluster ('<vc>', '</vc>'), that have been previously marked out in the parsing stage.

```
(11) define VCerror [ "<vc>" [VC - [VC1 |
                              VC2]] "</vc>" ];
```

By application of a marking transducer in (12), the found error segment is annotated directly in the text as in example (13).

```
(12) define markVCerror [VCerror ->
          "<Error verb after Vaux>" ... "</Error>"];
```

(13) Men <vp> <vpHead> kom ihåg </vpHead> </vp> att <np> det </np> <vp> <vpHead> inte **<Error verb after Vaux>** <vc> ska blir </vc> **</Error>** </vpHead> <np> någon <ap> riktig </ap> brand </np> </vp>

5 Parsing

5.1 Parsing procedure

The rules of the (underspecified) *broad grammar* are used to mark syntactic patterns in a text. A *partial, lexical-prefix-first, longest-match, incremental* strategy is used for parsing. The parsing procedure is *partial* in the sense that only portions of text are recognized and no full parse is provided for. Patterns not recognized by the rules of the (*broad*) grammar remain unchanged. The maximal instances of a particular phrase are selected by application of the *left-to-right-longest-match replacement* operator.

The segments are built on in cascades in the sense that first the heads are recognized, starting from the left-most edge to the head (so called *lexical-prefix*) and then the segments are expanded in the next level by addition of complement constituents. The regular expressions in (14) compose the marking transducers of separate segments into a three step process.

```
(14)    define parse1[markVPhead .o.
                  markPPhead  .o. AP];
        define parse2 [markNP];
        define parse3 [markPP .o. markVP];
```

First the verbal heads, prepositional heads and adjective phrases are recognized by composition in that order (*parse1*). This output serves then as input to the next level, where the adjective phrases are extended and noun phrases are recognized and marked (*parse2*). This output in turn serves as input to the last level, where the whole prepositional phrases and verb phrases are recognized in that order (*parse3*). During and after this parsing annotation, some phrase types are further expanded with post-modifiers, split segments are joined and empty results are removed.

The 'broadness' of the grammar and the lexical ambiguity in words, necessary for parsing text containing errors, also yields ambiguous and/or alternative phrase annotations. We block some of the (erroneous) alternative parses by the order in which phrase segments are selected, which causes bleeding of some rules (i.e. the parsing order destroys application of another parsing rules; a feature mostly used of the ordering of phonological rules) and more 'correct' parsing results are achieved. The order in which the labels are inserted into the string influences the segmentation of patterns into phrases. Further ambiguity resolution is provided for by filtering automata.

5.2 The Heuristics of Parsing Order

Reordering rules used in parsing allows us to resolve certain ambiguities. For example, marking verbal heads before noun phrases will prefer a verb phrase interpretation of a string over a noun phrase interpretation and avoid merging constituents of verbal heads into noun phrases and yielding noun phrases with too-wide range.

For instance, marking first the sentence in (15) for noun phrases will interpret the pronoun *De* 'they' as a determiner and the verb *såg* 'saw', that is exactly as in English homonymous with the noun 'saw', as a noun and merges these two constituents to a noun phrase as shown in (16). *De såg* will subsequently be marked as ungrammatical, since a number feature mismatch occurs between the plural *De* 'they' and singular *såg* 'saw'.

(15) *De såg ledsna ut*
they looked sad out
- They seemed sad.

(16) <np>De såg </np> <np>ledsna </np> ut .

Composing the marking transducers by first marking the verbal head and then the noun phrase will instead yield the more correct parse. Although the alternative of the verb being parsed as verbal head or a noun remains (i.e. *såg* 'saw' is still tagged as a noun in a noun phrase), the pronoun *De* 'they' is now marked correctly as a separate noun phrase and not merged together with the main verb into a noun phrase:

(17) <np> De </np> <vpHead> <np> såg </np> </vpHead> <np> ledsna </np> ut .

The output at this stage is then further refined and/or revised by application of *filtering* transducers. Earlier parsing decisions depending on lexical ambiguity are resolved (e.g. adjectives parsed as verbs) and phrases extended (e.g. with postnominal modifiers). Other structural ambiguities, such as verb coordinations or clausal modifiers on nouns, are also taken care of.

This ordering strategy is not absolute however, since the opposite scenario is possible where parsing noun phrases before verbal heads is more suitable, as for instance in example (18) below, where the string *det öppna fönstret* 'the open window' will be split in three separate noun phrase segments when applying the order of parsing verbal heads before noun phrases, due the homonymity between an adjective and an infinitive or imperative verb form (19).

(18) *han tittade genom det öppna fönstret*
he looked through the open window
- He looked through the open window

(19) <np> han </np><vpHead> tittade </vpHead> genom <np> det </np> <vpHead> <np> öppna </np> </vpHead> <np> fönstret </np>

We analyzed the ambiguity frequency in the *Child Data* corpus and found that occurrences of nouns recognized as verbs are more frequent than the opposite. On this ground, we chose the strategy of marking verbal heads before marking noun phrases. In the case of the opposite scenario, the false parsing can be revised and corrected by an additional filter (see Section 5.3).

A similar problem occurs with homonymous prepositions and nouns. For instance, the string *vid* is ambiguous between an adjective ('wide') and a preposition ('by') as shown in example (20) and influences the order of marking prepositional heads and noun phrases. Parsing prepositional heads before noun phrases is more suitable for preposition occurrences as shown in (22) in order to prevent the preposition from being merged as part of a noun phrase, as in (21):

(20) *Jag satte mig vid bordet*
I sat me by the-table
– I sat down at the table.

(21) <np> Jag </np> satte <np> mig </np> <np> <ppHead> vid </ppHead> bordet </np>

(22) <np> Jag </np> satte <np> mig </np> <ppHead> <np> vid </np> </ppHead> <np> bordet </np>

5.3 Further Ambiguity Resolution

Nouns, adjectives and pronouns are homonymous with verbs and might then be interpreted by the parser as verbal heads. Adjectives homonymous with prepositions can be analyzed as prepositional heads. These parsing decisions can be redefined at a later stage by application of *filtering* transducers.

As exemplified in (19) above, the consequence of parsing verbal heads before noun phrases may yield noun phrases that are split into parts, due to the fact that adjectives are interpreted as verbs. The filtering transducer in (23) adjusts such segments and removes the erroneous (inner) syntactic tags (i.e. replaces them with the empty string '0') so that only the outer noun phrase markers remain and converts the split phrase in to one noun phrase yielding (24).

```
(23) define adjustNPAdj [
        "</np><vpHead><np>" -> 0 || Det _ APPhr
        "</np></vpHead>" NPPhr,,
        "</np></vpHead><np>"  ->  0  ||  Det
        "</np><vpHead><np>" APPhr _ ];
```

(24) <np> han </np> <vpHead> tittade </vpHead> genom <np> det öppna fönstret </np>

The regular expression consists of two replacement rules that apply in parallel. They are constrained by the surrounding context of a preceding determiner (Det) and a subsequent adjective phrase (APPhr) and a noun phrase (NPPhr) in the first rule, and a preceding determiner and an adjective phrase in the second rule.

5.4 Parsing Expansion and Adjustment

The text is now annotated with syntactic tags and some of the segments have to be further expanded with *postnominal attributes* and *coordinations*. In the current system, partitive prepositional phrases are the only postnominal attributes taken care of. The reason is that grammatical errors were found in these constructions.

By application of the filtering transducer in (25) the example text in (26) with the partitive noun phrase *en av dom gamla husen* 'one of the old houses' split into a noun phrase followed by a prepositional head that includes the partitive preposition *av* 'of' and yet another noun phrase from the parsing stage (27) is merged to form a single noun phrase, as shown in (28). This automaton removes the redundant inner syntactic markers by application of two replacement rules, constrained by the right or left context. The replacement occurs simultaneously by application of parallel replacement.

(25) define adjustNPPart [
"</np><ppHead>" -> 0 || _ PPart
"</ppHead><np>",,
"</ppHead><np>" -> 0 ||
"</np><ppHead>" PPart _];

(26) *Virginia hade öppnat en tygaffär i en av dom gamla husen.*
Virginia had opened a fabric-store in one of the old houses[def].
- Virginia had opened a fabric-store in one of the old houses.

(27) <np> Virginia </np> <vp><vpHead> <vc> hade öppnat </vc> </vpHead> <np> en tygaffär </np> i <np> en </np> <ppHead> av </ppHead> <np> dom <ap> gamla </ap> husen </np> .

(28) <np> Virginia </np> <vp> <vpHead> <vc> hade öppnat </vc> </vpHead> <np> en tygaffär </np> i <NPPart> en av dom <ap> gamla </ap> husen </np>

Other filtering transducers are used for refining the parsing result and eliminate incomplete parsing decisions such as prepositional heads without a following noun phrase.

6 The System Performance

6.1 Result on Child Data

The implemented error detector, *FiniteCheck*, cannot at present be considered as a fully developed grammar checking tool, but still even with its restricted lexicon and small grammar the results are promising. So far the technique was used to detect agreement errors in noun phrases, selection of finite and non-finite verb forms in main and subordinate clauses and infinitival complements. The implementation proceeded in two steps. In the first phase we devoted all effort to detection of the grammar errors, working mostly with the errors and not paying much attention to the text as a whole. The second phase involved blocking of the resultant false alarms found in the first stage.

In *Table 1* we show the final results of error detection in the training corpus of Child Data. There were altogether 15 agreement errors in noun phrase, 110 errors in the form of finite verb, 7 errors in the verb form after an auxiliary verb and 4 errors in verbs after infinitive marker.

Error type	No. Errors	CA	FA	R	P	F
Agreement in NP	15	15	62	100%	19%	33%
Finite verb form	110	96	126	87%	43%	58%
Verb form after aux. verb	7	6	47	86%	11%	20%
Verb form after inf. marker	4	4	0	100%	100%	100%
Total	136	121	235	89%	34%	49%

Table 1. Performance of *FiniteCheck* on Child Data: correct alarms (CA), false alarms (FA), recall (R), precision (P), F-value (F).

FiniteCheck detected all the agreement errors in noun phrases and all erroneous verb forms after an infinitive marker, only a portion of other errors in verb form was missed. The precision of the system is rather low, primarily due the ambiguity of the texts and the number of alarms marking other errors such as segmentation or spelling errors. This side-effect is difficult to eliminate totally and gives rather rise to new questions of how to handle also these types of writing problems that concern spelling rather than grammar.

The three Swedish grammar checkers mentioned above in Section 2: *Grammatifix*, *Granska* and *Scarrie*, have been tested on the *Child Data*. The result of their performance is shown in *Figure 2*, below, together with the results of *FiniteCheck*.

These three tools are designed to detect errors in text different from the nature of the *Child Data* and thus not surprisingly the accuracy rates are in overall low. The total recall rate for the four error types covered by *FiniteCheck* is between 9% and 21% in these three tools and precision varies between 16% to 35%. Errors in noun phrases seem to be better covered than verb errors.

In the case of *Grammatifix*, errors in verbs are not covered at all. Half of the noun phrase errors were identified and only five errors in the finite verb form. *Granska* covered all four error types and detected at most half of the errors for three of these types. However, only seven instances of errors in finite verb form were identified. *Scarrie* had difficulties with errors in verb form after infinitive marker that were not detected at all. Errors in noun phrase were the best detected type.

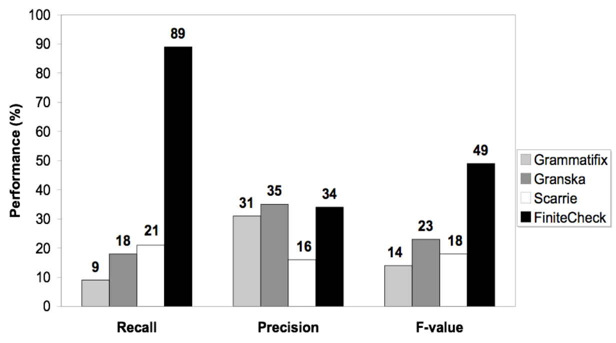


Figure 2: Performance of All Systems on Child Data

The detection performance of these three tools on Child Data is in general half that good in comparison to our detector and the fact that the error type with worst coverage (finite verbs) is the one most frequent among children indicates clearly the need for specialized grammar checking tools for children.

6.2 Result on Text Written by Adult

The current system was also tested on a text of 1 070 words written by an adult, one of the demonstration texts used by *Granska*. The performance of *FiniteCheck* on this text is presented in *Table 2*. We found 17 noun phrase agreement errors, 5 errors in the form of finite verb and 1 error in the verbform after an auxiliary verb in the text. *FiniteCheck* found all the verb form errors and most of the agreement errors, ending in a recall value of 87%. False alarms occurred also only in the agreement errors, resulting in a precision rate of 71% and an F-value of 78%.

Error type	No. Errors	CA	FA	R	P	F
Agreement in NP	17	14	6	82%	70%	76%
Finite verb form	5	5	1	100%	83%	91%
Verb form after aux. verb	1	1	1	100%	50%	67%
Total	23	20	8	87%	71%	78%

Table 2. Performance of *FiniteCheck* on Text Written by Adult: correct alarms (CA), false alarms (FA), recall (R), precision (P), F-value (F).

The three Swedish grammar checkers were also tested on this adult text, that reflects more the text type these tools are designed for. The results presented in *Figure 3* show an average recall rate of 52% for the three Swedish grammar checkers, *FiniteCheck* scored 87%. These tools had difficulties to detect the verb form errors, whereas most of the errors in noun phrase agreement were found. The opposite scenario applies for precision, where *FiniteCheck* had slightly worse rate (71%) than *Grammatifix* and *Granska*, which had a precision above 90%. *Scarrie's* precision was 65%. In the combined measure of recall and precision (F-value) our system obtained 78%, which is slightly better in comparison to the other tools that had 70% or less in F-value.

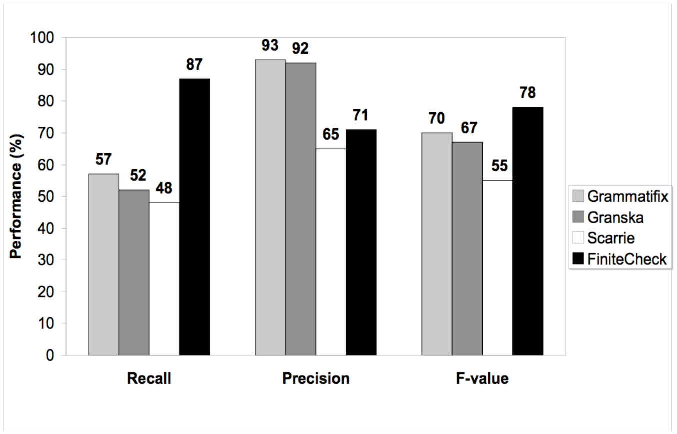


Figure 3: Performance of All Systems on Text Written by Adult

7 Conclusion

The simple finite state technique of subtraction presented in this paper, has the advantage that the grammars one needs to write to find errors are always positive grammars rather than grammars written to find specific errors. Thus, covering the valid rules of language means that the rule sets remain quite small and practically no prediction of errors is necessary.

The approach aimed further at minimal information loss in order to be able to handle text containing errors. The degree of ambiguity is maximal at the lexical level, where we choose to attach all lexical tags to strings. At higher levels, structural ambiguity is treated by parsing order, grammar extension and some other heuristics. There is an essential problem of ambiguity resolution on complement decisions that remains to be solved. Sequences of words grammatical in one context and ungrammatical in another are treated the same. The system overinterprets and gives rise to false alarms, mostly due the application of longest-match, but more seriously information indicating an error may be filtered out by erroneous segmentation and errors overlooked. A 'higher' mechanism is needed in order to solve these problems that takes into consideration the complement distribution and solves these structural dependencies.

The linguistic accuracy of the system is comparable to other Swedish grammar checking tools, that actually performed worse on the Child Data. The low performance of the Swedish tools on Child Data motivates clearly the need for adaptation of grammar checking techniques to children. The other tools obtained in general much lower recall values and although the error type of particular error was defined, the systems had difficulties to identify the errors, probably due problems to handle such a disrupted structure with many adjoined sentences and high error frequency.

Further, the robustness and modularity of this system makes it possible to perform both error detection and diagnostics and that the grammars can be reused for other applications that do not necessarily have anything to do with error detection, e. g. for educational purposes, speech recognition, and for other users such as dyslectics, aphasics, deaf and foreign speakers.

References

Ait-Mohtar, Salah and Chanod, Jean-Pierre (1997) Incremental Finite-State Parsing, In *ANLP'97,* Washington, pp. 72-79.

Arppe, A. (2000) Developing a Grammar Checker for Swedish.In *The 12th Nordic Conference of Computational Linguistics,* NODALIDA-99, pp.13-27.

Birn, J. (1998). *Swedish Constraint Grammar: A Short Presentation.* [http://www.lingsoft.fi/doc/swecg/].

Birn, J. (2000) Detecting Grammar Errors with Lingsoft's Swedish Grammar Checker. In *The 12th Nordic Conference of Computational Linguistics,* NODALIDA-99, pp.28-40.

Bustamente, F. R. and León, F. S. (1996) GramCheck: A Grammar and Style Checker. In *The 16th International Conference on Computational Linguistics*, Copenhagen, pp. 175-181.

Carlberger, J. and Kann, V. (1999) Implementing an efficient part-f-speech tagger. *Software – Practice and Experience,* 29(9):815-832.

Chanod, J.-P. (1996) A Broad-Coverage French Grammar Checker: Some Underlying Principles. In *The Sixth International Conference on Symbolic and Logical Computing,* Dakota State University Madison, South Dakota.

Domeij, R. (2003). *Datorstödd språkgranskning under skrivprocessen. Svensk språkkontroll ur användarperspektiv.* Doktorsavhandling, Stockholms Universitet, Institutionen för lingvistik.

Ejerhed, E., Källgren, G., Wennstedt, O. and Åström, M. (1992) *The Linguistic Annotation System of the Stockholm-Umeå Corpus Project.* Report 33. University of Umeå, Department of Linguistics.

Heidorn, G. (1993). Experience with an easily computed metric for ranking alternative parses. In Jensen, K., Heidorn, G., and Richardson, S. D. (eds.) *Natural Language Processing: The PLNLP Approach*. Kluwer Academic Publishers, Dordrecht.

Hård af Segerstad, Y. and Sofkova Hashemi, S. (2006) Learning to Write in the Information Age: A Case Study of Schoolchildren's Writing in Sweden. In Van Waes, L., Leijten, M. och Neuwirth, C. (eds.), *Writing and Digital Media*, Elsevier.

Jensen, K., Heidorn, G. and Richardsson, S. D. (eds.) (1993) *Natural Language Processing: The PLNLP Approach.* Kluwer Academic Publishers, Dordtrecht.

Järvinen, T. and Tapanainen, P. (1998). Towards an implementable dependency grammar. In Kahane, S. and Polguere, A. (eds.) The *Proceedings of COLIN-*

GACL'98, Workshop on 'Processing of Dependency-Based Grammars', pages 1–10. Universite de Montreal, Canada.

Karlsson, F. (1992). SWETWOL: Comprehensive morphological analyzer for Swedish. *Nordic Journal of Linguistics*, 15:1–45.

Karlsson, F., Voutilainen, A., Heikkilä, J., and Anttila, A. (1995). *Constraint Grammar: a language-independent system for parsing unrestricted text.* Mouton de Gruyter, Berlin.

Karttunen, L. (1993) *Finite State Lexicon Compiler.* Technical Report ISTL-NLTT-1993-04-02, Xerox Palo Alto Research Center, Palo Alto, California.

Karttunen, L., Chanod, J., Grefenstette, G. and Schiller, A. (1996) Regular Expressions for Language Engineering, In *Natural Language Engineering 2 (4)* 305-328.

Karttunen, L., Gaál, T. and Kempe, A. (1997) *Xerox Finite State Tool*. Xerox Research Centre Europe,

Kirschner, Z. (1994) CZECKER -a Maquette Grammar-Checker for Czech. In *The Prague Bulletin of Mathematical Linguistics 62,* Praha: Universita Karlova.

Knutsson, O. (2001). *Automatisk språkgranskning av svensk text.* Licentiatavhandling, KTH, Institutionen för numerisk analys och datalogi, Stockholm.

Kukich, K. (1992) Techniques for Automatically Correcting Words in Text. *ACM Computing Surveys*, Vol. 24, No. 4: 377 - 439.

Sofkova Hashemi, S. (2003) *Automatic Detection of Grammar Errors in Primary School Children's Texts. A Finite State Approach.* Doctoral dissertation. Gothenburg Monographs in Linguistics 24. Department of Linguistics, Göteborg University.

Sofkova Hashemi, S. (forthcoming) *The role of writing aid in the text production of school children*. Department of Linguistics, Göteborg University

Sågvall Hein, A. (1983). *A Parser for Swedish. Status Report for SveUcp*. (UCDLR-83-2). Uppsala University, Department of Linguistics. February 1983.

Sågvall Hein, A. (1999) *A grammar checking module for Swedish.* Report from the Scarrie-project: DEL 6.6.3, June 1999, Dept. of Linguistics, Uppsala University.

Sågvall Hein, A., Olsson, L.-G., Dahlqvist, B., and Mats, E. (1999). Evaluation report for the Swedish prototype. In Sågvall Hein, A. (ed.) *Reports from the SCARRIE project,* Deliverable 8.1.3, June 1999. Uppsala University, Department of Linguistics.

Vernon, A. (2000) Computerized grammar checkers 2000: Capabilities, limitations, and pedagogical possibilities. *Computers and Composition* 17, 329-349.

Vosse, T. G. (1994) *The Word Connection. Grammar-based Spelling Error Correction in Dutch.* Enschede: Neslia Paniculata.

Wedbjer Rambell, O. (1999). Swedish phrase constituent rules. A formalism for the expression of local error rules for Swedish. In Sågvall Hein, A. (ed.) *Reports from the SCARRIE project.* Uppsala University, Department of Linguistics.

Nbest Dependency Parsing with linguistically rich models

Xiaodong Shi
Institute of Artificial Intelligence
Department of Computer Science
Xiamen University, Xiamen 361005
mandel@xmu.edu.cn

Yidong Chen
Institute of Artificial Intelligence
Department of Computer Science
Xiamen University, Xiamen 361005
ydchen@xmu.edu.cn

Abstract

We try to improve the classifier-based deterministic dependency parsing in two ways: by introducing a better search method based on a non-deterministic nbest algorithm and by devising a series of linguistically richer models. It is experimentally shown on a ConLL 2007 shared task that this results in a system with higher performance while still keeping it simple enough for an efficient implementation.

1 Introduction

This work tries to improve the deterministic dependency parsing paradigm introduced in (Covington 2001, Nivre 2003, Nivre and Hall, 2005) where parsing is performed incrementally in a strict left-to-right order and a machine learned classifier is used to predict deterministically the next parser action. Although this approach is very simple, it achieved the state-of-art parsing accuracy. However, there are still some problems that leave further room for improvement:

(1) A greedy algorithm without backtracking cannot ensure to find the optimal solution. In the course of left-to-right parsing, when further context is seen, the previous decisions may be wrong but a deterministic parser cannot correct it. The usual way of preventing early error "commitment" is to enable a k-best or beam-search strategy (Huang and Chiang 2005, Sagae and Lavie 2006).

(2) A classifier based approach (e.g. using SVM or memory based learning) is usually linguistically naïve, to make it applicable to multiple languages. However, a few studies (Collins 1999, Charniak et al 2003, Galley et al 2006) have shown that linguistically sophisticated models can have a better accuracy at parsing, language modeling, and machine translation, among others.

In this paper we explore ways to improve on the above-mentioned deterministic parsing model to overcome the two problems. The rest of the paper is organized as follows. Section 2 argues for a search strategy better at finding the optimal solution. In section 3 we built a series of linguistically richer models and show experimental results demonstrating their practical consequences. Finally we draw our conclusions and point out areas to be explored further.

2 Dependency Parsing Enhancements

In the classifier-based approach as in Nivre (2003) a parse tree is produced by a series of actions similar to a left-to-right shift-reduce parser. The main source of errors in this method is the irrevocability of the parsing action and a wrong decision can therefore lead to further inaccuracies in later stages. So it cannot usually handle garden-path sentences. Moreover, each action is usually predicted using only the local features of the words in a limited window, although dynamic features of the local context can be exploited (Carreras 2006).

To remedy this situation, we just add a scoring function and a priority queue which records nbest partial parses. The scoring function is defined on the parsing actions and the features of a partial parse. It can be decomposed into two subfunctions:

score(a,y)=parsing_cost(a,y) + lm(y)

where **a** is parsing actions and **y** is partial parses, and parsing cost (*parsing_cost*) is used to implement certain parsing preferences while the lingustic model score (*lm*) is usually modeled in the linguistic (in our case, dependency model) framework.

In the usual nbest or beam-search implementation (e.g. Huang and Chiang 2005, Sagae and Lavie 2006), only *lm* is present.

We give justification of the first term as follows: Many probability functions need to know the dependency label and relative distance between the dependent and the head. However, during parsing sometimes this head-binding can be very late. This means a right-headed word may need to wait very long for its *right* head, and so a big partial-parse queue is needed, while psychological evidence suggests that there is some overhead involved in processing every word and a word tends to attach locally. By modeling parsing cost we can first use a coarse probability model to guide the nbest partial results in order not to defer the probability calculation. As parsing progresses, more information becomes available; we can have a better estimation of our linguistic probability model to rectify the inaccuracy.

This use of a coarse scoring mechanism to guide the early parsing for possible later rectification of the decision is a novel feature of our parsing framework and enables better searching of the solution space. To implement it, we just remember the exact score of the every major decision (wait, add a dependent or attach a head) in parsing, and re-score when more context is available. Compared with (Charniak 2005), our parsing process requires only one pass.

Thus, we can strike a balance between accuracy, memory and speed. With a moderately-sized *n* (best partial results), we can reduce memory use and get higher speed to get a same accuracy. An added advantage is that this idea is also useful in other bottom-up parsing paradigms (not only in a dependency framework).

In a word, our main innovation is the use of a parsing cost to influence the search paths, and the use of an evolving *lm* function to enable progressively better modeling. The nbest framework is general enough to make this a very simple modification to the basic algorithm of Nivre (2003).

3 Better Linguistic Modeling

In our modeling we combine different linguistic models by using many probability functions:

$$\mathrm{lm}(y) = \Sigma \log P(w_i, w_j, x, y) = \Sigma\, \mathbf{W} * \log \mathbf{P}$$

where **w** are the trained weight vector and **P** is a vector of probability functions. In our system we considered the following functions:

P1: function measuring the probability of a head and a dependent. This is the base function in most dependency parsing framework.

P2: function calculating the subcategorization frame probability;

P3: function calculating the semantic frame using a Chinese FrameNet (Liu 2006).

P4: function measuring the semantic affinity between a head and a dependent using resources such as Hownet (Dong 2003).

P5: Other Chinese specific probability functions defined on the features of the head, the dependents, the partial parse and the input.

Model P2 is a probability function on *pseudo* subcategorization frames (as a concatenation of all the dependents' labels) as we don't know the distinction of arguments and adjuncts in the dependency Treebank. We used a Markovian subcategorization scheme with left and right STOP delimiters to ease the data sparseness. And as a first approximation, we also experimented with a model where each label can only be used a certain times in a direction. This model is called P2' in Table 4.

Other functions (P3-P5) are also very useful with its different linguistic content. Model P5 actually contains a lot of Chinese-specific functions, e.g. between a sentence-final particle and a verb.

We designed a series of experiments to show to effectiveness of each model. We use the Chinese training data of the ConLL 2007 shared task. We divided the training data by a 9:1 split. Table 1 shows the statistics.

	Training	testing
sentences	51777	5180
Words	302943	34232

Table 1. Experimental data

In the baseline model, we train a simple probability function between a head and a dependent using deleted interpolation. For nbest=1, we have a deterministic model.

	LAS	UAS	time
Deterministic	41.64 %	44.11 %	8s
nbest = 50	71.30 %	76.34 %	72s
nbest = 500	71.90 %	76.99 %	827s

Table 2. baseline systems

It can be seen (Table 3) that combing different linguistic information can lead to significant increase of the accuracy. However, different models have different contributions. Our experiments confirm with Collins's result in that subcategorization carries very important linguistic content.

	LAS	UAS	time
P1	71.90 %	76.99 %	827s
P1 + P2'	73.45 %	78.44 %	832s
P1 + P2' + P2	77.92 %	82.42 %	855s
P1 + P2 + P3	79.13%	83.57%	1003s
P1-4	81.21%	85.78%	1597s
P1-5	83.12%	87.03%	2100s
Verb valency	85.32 %	89.12 %	-
DE refinement	85.98%	90.20%	-

Table 3. systems with different linguistic models

3.1 Relabeling of the parse treebank

Sometimes the information needed in the modeling is not in the data explicitly. Implicit information can be made explicit and accessible to the parser.

In the Chinese Treebank the relation label is often determined by the head word's semantic type. We tried the relabeling of coarse POS info of the verb in a effort to detect its valency; and refinement of the auxiliary word 的 DE (as error analysis shows it is the where the most errors occur). Results are in Table 3.

We also tried refinement of the relation label by using the two connected words. However, this does not improve the result. Automatic linguistic modeling using latent label (Matsuzaki 2005) can also be attempted but is not yet done.

4 Conclusions

In this paper we showed that simple classifier-based deterministic dependency parsing can be improved using a more flexible search strategy over an nbest parsing framework and a variety of linguistically richer models. By incorporating different linguistic knowledge, the parsing model can be made more accurate and thus achieves better results.

Further work to be done includes ways to combine machine learning based on the automatic feature selection with manual linguistic modeling: an interactive approach for better synergistic modeling (where the machine proposes and the human guides). Various a priori models can be tried by the machine and patterns inherent in the data can be revealed to the human who can then explore more complex models.

References

Xavier Carreras, Mihai Surdeanu, and Lluís Màrquez. 2006. *Projective Dependency Parsing with Perceptron*. In Proceedings of CoNLL-X. 181-185.

Liang Huang and David Chiang. 2005. Better *k*-best parsing. In *Proceedings of IWPT*.

Eugene Charniak; K. Knight, and K.Yamada. 2003. *Syntax-based language models for statistical machine translation*. In MT Summit IX. Intl. Assoc. for Machine Translation.

Eugene Charniak and Mark Johnson. 2005. *Coarse-tofine n-best parsing and maxent discriminative reranking*. In Proceedings of ACL.

Michael Collins. 1999. *Head-Driven Statistical Models-for Natural Language Parsing*. PhD Dissertation, University of Pennsylvania.

Michael Collins. 2004. Parameter Estimation for Statistical Parsing Models: Theory and Practice of Distribution-Free Methods. In Harry Bunt el al, *New Developments in Parsing Technology*, Kluwer.

Michael A. Covington. 2001. *A fundamental algorithm for dependency parsing*. Proceedings of the 39th Annual ACM Southeast Conference, pp. 95-102.

Zhendong Dong and Qiang Dong. 2003. *HowNet - a hybrid language and knowledge resource*. In Proceeding of Natural Language Processing and Knowledge Engineering.

M. Galley, J. Graehl, K. Knight, D. Marcu, S. DeNeefe, W. Wang, and I. Thayer. 2006. *Scalable Inference and Training of Context-Rich Syntactic Models*. In Proc. ACL-COLING.

Kaiying Liu. 2006. *Building a Chinese FrameNet*. In Proceeding of 25th anniversary of Chinese Information Processing Society of China.

Takuya Matsuzaki, Yusuke Miyao, Jun'ichi Tsujii. 2005. *Probabilistic CFG with latent annotations*. In Proceedings of ACL-2005.

Joakim Nivre. 2003. *An efficient algorithm for projective dependency parsing*. In Proceedings of IWPT. 149-160.

Joakim Nivre and Johan Hall. 2005. *MaltParser: A Language-Independent System for Data-Driven Dependency Parsing*. In Proceedings of the Fourth Workshop on Treebanks and Linguistic. Theories, Barcelona, 9-10 December 2005. 137-148.

Sagae, K. and Lavie, A. 2006 *A best-first probabilistic shift-reduce parser*. In Proceedings of ACL.

Symbolic Preference Using Simple Scoring

Paula S. Newman
newmanp@acm.org

Abstract

Despite the popularity of stochastic parsers, symbolic parsing still has some advantages, but is not practical without an effective mechanism for selecting among alternative analyses. This paper describes the symbolic preference system of a hybrid parser that combines a shallow parser with an overlay parser that builds on the chunks. The hybrid currently equals or exceeds most stochastic parsers in speed and is approaching them in accuracy. The preference system is novel in using a simple, three-valued scoring method (-1, 0, or +1) for assigning preferences to constituents viewed in the context of their containing constituents. The approach addresses problems associated with earlier preference systems, and has considerably facilitated development. It is ultimately based on viewing preference scoring as an engineering mechanism, and only indirectly related to cognitive principles or corpus-based frequencies.

1 Introduction

Despite the popularity of stochastic parsers, symbolic parsing still has some advantages, but is not practical without an effective mechanism for selecting among alternative analyses. Without it, accept/fail grammar rules must either be overly strong or admit very large numbers of parses. .

Symbolic parsers have recently been augmented by stochastic post-processors for output disambiguation, which reduces their independence from corpora. Both the LFG XLE parser (Kaplan et.al. 2004), and the HPSG LinGO ERG parser (Toutanova et al. 2005) have such additions.

This paper examines significant aspects of a purely symbolic alternative: the preference and pruning system of the RH (Retro-Hybrid) parser (Newman, 2007). The parser combines a pre-existing, efficient shallow parser with an overlay parser that builds on the emitted chunks. The overlay parser is "retro" in that the grammar is related to ATNs (Augmented Transition Networks) originated by Woods (1970).

RH delivers single "best" parses providing syntactic categories, syntactic functions, head features, and other information (Figure 1). The parenthesized numbers following the category labels in the figure are preference scores, and are explained further on. While the parses are not quite as detailed as those obtained using "deep" grammars, the missing information, mostly relating to long distance dependencies, can be added at far less cost in a post-parse phase that operates only on a single best parse. Methods for doing so, for stochastic parser output, are described by Johnson (2002) and Cahill et al (2004).

The hybrid parser exceeds most stochastic parsers in speed, and approaches them in accuracy, even based on limited manual "training" on a particular idiom, so the preference system is a successful one (see Section 6), and continues to improve.

The RH preference system builds on earlier methods. The major difference is a far simpler scoring system, which has considerably facilitated overlay parser development. Also, the architecture allows the use of large numbers of preference tests without impacting parser speed. Finally, the treatment of coordination exploits the lookaheads afforded by the shallow parser to license or bar alternative appositive readings.

Section 2 below discusses symbolic preference systems in general, and section 3 provides an overview of RH parser structure. Section 4 describes the organization of the RH preference system and the simplified scoring mechanism. Section 5 discusses the training approach and Section 6 provides some experimental results. Section 7 summarizes, and indicates directions for further work.

Proceedings of the 10th Conference on Parsing Technologies, pages 83–92,
Prague, Czech Republic, June 2007. ©2007 Association for Computational Linguistics

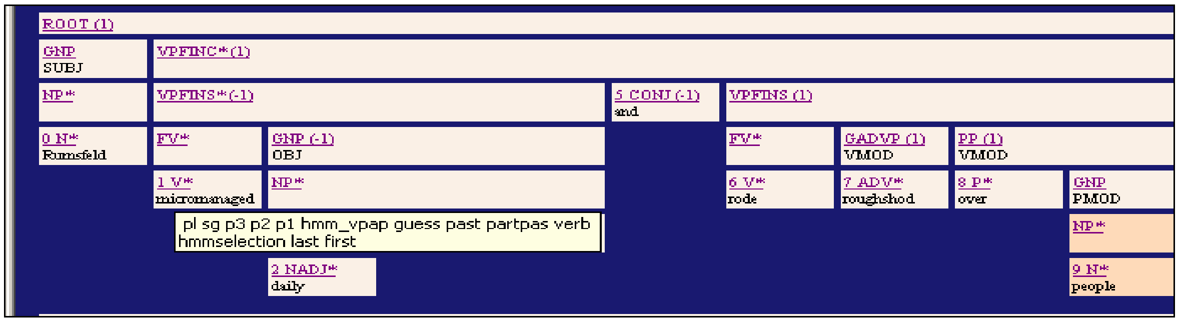


Figure 1. Output Parse Tree for *"Rumsfeld micromanaged daily briefings and rode roughshod over people."* * indicates head. Mouseover shows head features for "micromanaged".

2 Background: Symbolic Preference

2.1 Principles

Preference-based parsing balances necessarily permissive syntactic rules by preference rules that promote more likely interpretations. One of the earliest works in the area is by Wilks (1975), which presented a view of preference as based on semantic templates. Throughout the 1980's there was a considerable amount of work devoted to finding general principles, often cognitively oriented, for preference rules, and then to devise mechanisms for using them in practical systems. Hobbs and Bear (1990) provide a useful summary of the evolved principles. Slightly restated, these principles are:

1. Prefer attachments in the "most restrictive context".
2. If that doesn't uniquely determine the result, attach low and parallel, and finally
3. Adjust the above based on considerations of punctuation

Principle 1 suggests that the preference for a constituent in a construction should depend on the extent to which the constituent meets a narrow set of expectations. Most of the examples given by Hobbs and Bear use either (a) sub-categorization information, e.g., preferring the attachment of a prepositional phrase to a head that expects that particular preposition, or (b) limited semantic information, for example, preferring the attachment of a time expression to an event noun.

Principle 2 implies that in the absence of coordination, attachment should be low, and in the presence of coordination, parallel constituents should be preferred. Principle 3 relates primarily to the effect of commas in modifying attachment preferences.

2.2 Implementations

Abstractly, symbolic preference systems can be thought of as regarding a set of possible parses as a collection of spanning trees over a network of potential relationships, with each edge having a numeric value, and attempting to find the highest scoring tree.[1]

However, for syntactic parsers, in contrast with dependency parsers, it is convenient to associate scores with constituents as they are built, for consistency with the parser structure, and to permit within-parse pruning. A basic model for a preference system assigns preference scores to rules. For a rule

$$C \rightarrow c_1, c_2, \ldots, c_n$$

the preference score $PS(CC)$ of a resultant constituent CC is the sum:

$$PS(cc_1) + PS(cc_2) + \;\; +PS(cc_n) \\ + TRS\,(C, cc_1, cc_2, \ldots, cc_n)$$

where $PS(cc_i)$ is the non-contexted score of constituent cc_i, and the total relationship score TRS is a value that assesses the relationships among the sibling constituents of CC. The computation of TRS depends on the parser approach. For a top-down parser, TRS may be the sum of contexted relationship scores CRS, for example:

$$TRS = CRS\,(cc_1|C) + CRS\,(cc_2|C, cc_1), + \\ CRS\,(cc_3|C, cc_1, cc_2) + \ldots.. \\ + CRS\,(c_n\,|C,\; cc_1,\ldots.cc_{n-1})$$

where each $CRS\,(cc_i|\_\,)$ evaluates cc_i in the context of the prior content of the constituent CC and the category C..

Few publications specify details of how preference scores are assigned and combined. For example, Hobbs and Bear (1990) say only that "When a

[1] The idea has also been used directly in stochastic parsers that consider all possible attachments, for example, by McDonald et al. (2005).

non-terminal node of a parse tree is constructed, it is given an initial score which is the sum of the scores of its child nodes. Various conditions are checked during the construction of the node and, as a result, a score of 20, 10, 3, -3, -10, or -20 may be added to the initial score."

McCord (1993), however, carefully describes how the elements of *TRS* are computed in his slot grammar system. Each element value is the sum of the results of up to 8 optional, typed tests, relating to structural, syntactic, and semantic conditions. One of these tests, relating to coordination, is a complex test involving 7 factors assessing parallelism.

2.3 Multi-Level Contexted Scoring

The scores assigned by symbolic preference systems to particular relationships or combinations usually indicate not just whether they are preferred or dispreferred, but to what degree. For example, a score of 1 might indicate that a relationship is good, and 2 that it is better.

Such multi-level scores create problems in tuning parsers to remove undesirable interactions, both in the grammar and the preference system. Even for interactions foreseen in advance, one must remember or find out the sizes of the preferences involved, to decide how to compensate. Yamabana et al. (1993) give as an example a bottom-up parser, where an S constituent with a transitive verb head but lacking an object is initially given a strong negative preference, but when it is discovered that the constituent actually functions as a relative clause, the appropriate compensation must be found. (Their solution uses a vector of preference scores, with the vector positions corresponding to specific types of preference features, together with an accumulator. It allows the content of vector elements to be erased based on subsequently discovered compensating features.)

For unforeseen interactions, for example when a review of parser outputs finds that the best parse is not given the highest preference score, multi-level contexted scoring requires complex tracing of the contribution of each score to the total, remembering at each point what the score should be, to determine the necessary adjustments.

A different sort of problem of multi-level scoring stems from the unavoidable incompleteness of information. For example, in Figure 1, the attachment of an object to the "guessed" verb "micro-managed" is dispreferred because the verb is not identified as transitive. Here, the correct reading survives because there are no higher scoring ones. But in some situations, if such a dispreference were given a large negative score, the parser could be forced into very odd readings not compensated for by other factors.

2.4 Corpus-Based Preference

In the early 1990's, the increasing availability and use of corpora, together with a sense that multi-level symbolic preference scores were based on ad-hoc judgments, led to experiments and systems that used empirical methods to obtain preference weights. Examples of this work include a system by Liu et al (1990), and experiments by Hindle and Rooth (1993), and Resnik and Hearst (1993).[2]

These efforts had mixed success, suggesting that while multi-level preference scores are problematic, integrating some corpus data does not solve the problems. In light of later developments, this might be expected. Full-scale contemporary stochastic parsers use a broad range of interacting features to obtain their fine-grained results; frequencies of particular relationships are just one aspect.

2.5 OT-based Preference

A more recent approach to symbolic preference adapts optimality theory to parser and generator preference. Optimality Theory (OT) was originally developed to explain phonological rules (Prince and Smolensky, 1993). In that use, potential rules are given one "optimality mark" for each constraint they violate. The marks, all implicitly negative, are ranked by level of severity. A best rule $\boldsymbol{R}$ is one for which (a) the most severe level of constraint violation $\boldsymbol{L}$ is $\leq$ the level violated by any other rule, and (b) if other rules also violate level $\boldsymbol{L}$ constraints, the number of such violations is $\geq$ the number of violations by $\boldsymbol{R}$.

As adapted for use in the XLE processor for LFG (Frank et al. 1998) optimality marks are associated with parser and generator outputs. Positive marks are added, and also labeled inter-mark positions within the optimality mark ranking. The labeled positions influence processor behavior. For generation, they are used to disprefer infelicitous strings accepted in a parse direction. And for pars-

[2] McCord (1993) also includes some corpus-based information, but to a very limited extent.

ing they can be used to disprefer (actually ignore) rarely-applicable rules, in order to reduce parse time (Kaplan et al, 2004).

However, because the optimality marks are global, a single dispreference can rule out an entire parse. To partially overcome this limitation, a further extension (see XLE Online Documentation) allows direct comparisons of alternative readings for the same input extent. A different optimality mark can be set for each reading, and the use of one such mark in the ranking can be conditioned on the presence of another particular mark for the same extent. For example, a conditional dispreference can be set for an adjunct reading if an argument reading also exists. The extension does not address more global interactions, and is said (Forst et al. 2005) to be used mostly as a pre-filter to limit the readings disambiguated by a follow-on stochastic process.

2.6 A Slightly Different View

A slightly different view of preference–based parsing is that the business of a preference system is to work in tandem with a permissive syntactic grammar, to manipulate outcomes.

The difference focuses on the pragmatic role of preference in coercing the parser. In this light, the principles of section 2.1 are guidelines for desired outcomes, not bases for judging the goodness of a relationship or setting preference values. Instead, preference values should be set based on their effectiveness in isolating best parses. Also, in this light, the utility of a preference system lies not only in its contribution to accuracy, but also in its software-engineering convenience. These considerations led to the simpler, more practical scoring system of the RH overlay parser, described in section 4 below, in which contexted preference scores *CRS* can have one of only 3 values, -1, 0, or +1.

3 Background: The RH Parser

The RH parser consists of three major components, outlined below: the shallow parser, a mediating "locator" phase, and the overlay parser.

3.1 Shallow Parser

The shallow parser used, XIP, was developed by XRCE (Xerox Research Center Europe). It is actually a full parser, whose per-sentence output consists of a single tree of basic chunks, together with identifications of (sometimes alternative) typed dependences among the chunk heads (Ait-Mokhtar et al. 2002, Gala 2004). But because the XIP dependency analysis for English was not mature at the time that work on RH began, and because a classic parse tree annotated by syntactic functions is more convenient for some applications, we focused on the output chunks.

XIP is astonishingly fast, contributing very little to parse times (about 20%). It consists of the XIP processor, plus grammars for a number of languages. The grammar for a particular language consists of:

(a) a finite-state lexicon producing alternative part-of-speech and morphological analyses for each token, together with bit-expressed subcategorization and control features, and (some) semantic features,
(b) a substitutable tagger identifying the most probable part of speech for each token, and
(c) sequentially applied rule sets that extend and modify lexical information, disambiguate tags, identify named entities and other multiwords, and produce output chunks and inter-chunk head dependences (the latter not used in the hybrid).

Work on the hybrid parser has included large scale extensions to the XIP English rule sets.

3.2 Locator phase

The locator phase accumulates and analyses some of the shallow parser results to expedite the grammar and preference tests of the overlay parser.

For preference tests, for any input position, the positions of important leftward and rightward tokens are identified. These "important" tokens include commas, and leftward phrase heads that might serve as alternative attachment points.

Special attention is given to coordination, a constant source of inefficiency and inaccuracy for all parsers. To limit this problem, an input string is divided into spans ending at coordinating conjunctions, and the chunks following a span are examined to determine what kinds of coordination might be present in the span. For example, if a chunk following a span *Sp* is a noun phrase, and there are no verbs in the input following that noun phrase, only noun phrase coordination is considered within *Sp***.** Also, with heuristic exceptions, the locator phase disallows searching for appositives within long sequences of noun and prepositional phrases ending with a coordinating conjunction.

3.3 Overlay Parser

The overlay parser uses a top-down grammar, expressed as a collection of ATN-like grammar networks. A recursive control mechanism traverses the grammar networks depth-first to build constituents. The labels on the grammar network arcs represent specialized categories, and are associated with tests that, if successful, either return a chunk or reinvoke the control to attempt to build a constituents for the category. The label-specific tests include both context-free tests, and tests taking into account the current context. For details see (Newman, 2007).

If an invocation of the control is successful, it returns an **output network** containing one or more paths, with each path representing an alternative sequence of immediate children of the constituent. An example output network is shown in figure 2. Each arc of the network references either a basic chunk, or a final state of a subordinate output network. Unlike the source grammar networks, the output networks do not contain cycles or converging arcs, so states represent unique paths.

The states contain both (a) information about material already encountered along the path, including syntactic functions and head features, and (b) a preference score for the path to that point. Thus the figure 2 network represents two alternative noun phrases, one represented by the path containing OS_0 and OS_1, and one containing OS_0, OS_1, and OS_2. State OS_2 contains the preference score (+1), because attaching a locative pp to a feature of the landscape is preferred.

From	To	Cat	Synfun	Reference
OS_o	OS_1	NP	HEAD	NPChunk (*The park*)
OS_1	OS_2	PP	NMOD	Final state of PP net for (*in Paris*)
States		**Score**	**Final?**	
OS_0		0	No	
OS_1		0	Yes	
OS_2		+1	Yes	

Figure 2. Output network for "The park in Paris"

Before an output network is returned from an invocation of the control mechanism, it is pruned to remove lower-scoring paths, and cached.

Output from the overlay parser is a single tree (Figure 1) derived from a highest scoring full path (i.e. final state) of a topmost output network. If there are several highest scoring paths, low attach considerations select a "best" one. The preference scores shown in Figure 1 in parentheses after the category labels are the scores at the succeeding states of the underlying output networks.

4 Preference System

Any path in an output network has the form:

$S_0, Ref_1, S_1, Ref_2, \ldots, S_{n-1}, Ref_n, S_n$

where S_i is a state, and Ref_i labels an arc, and references either a basic chunk, or a final state of another output network. A state S_i has total preference score $TPS(i)$ where:

- $TPS(0) = 0$
- $TPS(i),\ i>0 =$
  $TPS(i-1) + PS(Ref_i) + CRS(Ref_i)$
- $PS(Ref_i)$ is the non-contexted score of the constituent referenced by Ref_i, that is, the score at the referenced final state.
- $CRS(Ref_i)$ is the contexted score for Ref_i, in the context of the network category and the path ending at the previous state i-1.

For example, if Ref_i refers to a noun phrase considered a second object within a path, and the syntactic head along the path does not expect a second object, $CRS(Ref_i)$ might be (-1).

Each value *CRS* is limited to values in {-1, 0, +1}. Therefore, no judgment is needed to decide the degree to which a contexted reference is to be dispreferred or preferred. Also, if the desired parse result does not receive the highest overall score, it is relatively easy to trace the reason. Pruning (see below) can be disabled and all parses can be displayed, as in Figure 1, which shows the scores $TPS(i)$ in parentheses after the category labels for each Ref_i (with zero scores not shown). Then, if

$TPS(i) > (TPS(i-1) + PS(Ref_i))$

it is clear that the contexted reference is preferred. If multi-level contexted scoring were used instead, it would be necessary to determine whether the reference was preferred to exactly the right degree.

Test Block Type	**Length Independent?**	**Indexed By**
Coordinate	Y	Parent syncat
Subcat	Y	No index
FN1	Y	synfun
TAG1	Y	syncat
FN2	N	synfun
TAG2	N	syncat

Table 1. Preference Test Block Types

4.1 Preference test organization

To compute the contexted score *CRS* for a reference, relevant tests are applied until either (a) a score of -1 is obtained, which is used as *CRS* for the reference, or (b) the tests are exhausted. In the latter case, *CRS* is the higher of the values {0, +1} returned by any test.

For purposes of efficiency, the preference tests are divided into typed blocks, as shown in Table 1. At most one block of each type can be applied to a reference. Four of the blocks contain tests that are independent of referenced constituent length. They are applied at most once for a returned output network and the results are assumed for all paths. The other two blocks are length dependent.

Referring to Table 1, the length-independent coordinate tests are applied only to non-first siblings of coordinated constituents. The parent category indicates the type of constituents being coordinated and selects the appropriate test block. Tests in these blocks focus on the semantic consistency of a coordinated sibling with the first one.

Subcategorization tests are applied to prepositional, particle, and clausal dependents of the current head. These tests consist to a large extent of bit-vector implemented operations, comparing the expected dependent types of the head with lexical features of the prospective dependent. The tests are made somewhat more complex because of various exceptions, such as (a) temporal and locative phrases, and (b) the presence of a nearer potential head also expecting the dependent type.

The other test block types are selected and accessed either by the syntactic category or the syntactic function of the reference, depending on the focus of the test. The length-dependent tests include tests of noun-phrases within coordinations to determine whether post modifiers should be applied to the individual phrase or to the coordination as a whole.

The test blocks are expressed in procedural code. This has allowed the parser to be developed without advance prediction of the types of information needed for the tests, and also has contributed some efficiency. The blocks, usually short but occasionally long, generally consist of ordered (if-then-else) subtests.

4.2 Preference test scope

A contexted preference test can refer to material on three levels of the developing parse tree: (a) the syntactic category of the parent (available because of the top-down parser direction) (b) information about the current output network path, including head features, already-encountered syntactic functions, and a small collection of special-purpose information, and (c) information about the referenced constituent, specifically its head and a list of the immediately contained syntactic functions. The tests can also reference lookahead information furnished by the locator phase. This material is sufficient for most purposes. Limiting the kind of referenced information, particularly not permitting access to sibling constituents or deep elements of the referenced constituent, contributes to performance.

4.3 Pruning

Before an output network is completed, it is pruned to remove lower-scoring output network paths. Any path with the same length as another but with a lower score is pruned. Also, paths having other lengths but considerably lower preference scores than the best-scoring path are often pruned as well.

4.4 Usage Example

To illustrate how the simple scores and modular tests are used to detect and repair problems in the preference system, Figure 1 shows, as noted before, that the attachment of an object to the guessed verb "micromanaged" is dispreferred. In this case the probable reason is the lack of a transitive feature for the verb. To check this, we would look at the FN1 test block for OBJ and find that in fact the test assigns (-1) in this case. The required modification is best made by adding a transitive feature to guessed verbs.

But there is another problem here: the attachment of the pp "over people" is not given a positive preference. Checking the FN1 test block for

VMOD and the TAG1 test block for PP finds that there is in fact no subtest that prefers combinations of motion verbs and "over". While this doesn't cause trouble in the example, it could if there were a prior object in the verb phrase. A subtest or subcategorization feature could be added.

5 Training the Preference System

To obtain the preference system, an initial set of tests is identified, based primarily on subcategorization considerations, and then refined and extended based on manual "training" on large numbers of documents. Several problem situations result in changes to the system, besides random inspection of scores:

(a) the best parse identified is not the correct one, either because the correct parse is not the highest scoring one, or because another parse with the same score was considered "best" because of low-attach considerations.

(b) The best parse obtained is the correct one, but there are many other parses with the same score, suggesting a need for refinement, both to improve performance and to avoid errors in related circumstances when the correct parse does not "float" to the top.

(c) No parse is returned for an input, because of imposed space constraints, which indirectly control the amount of time that can be spent to obtain a parse.

In some cases the above problems can be solved by adjusting the base grammar, or by extending lexical information to obtain the appropriate preferences. For example, the preference scoring problems of Figure 1 can be corrected by adding subcategorization information, as described above.

In other cases, one or more modifications to the preference system are made, adding positive tests to better distinguish best parses, adding negative tests to disprefer incorrect parses, and/or refining existing tests to narrow or expand applicability.

Positive tests often just give credit to expected structures not previously considered to require recognition beyond acceptance by the grammar. Negative tests fall into many classes, such as:

(a) Tests for "ungrammatical" phenomena that should not be ruled out entirely by the grammar. These include lack of agreement, lack of expected punctuation, and presence of unexpected punctuation (such as a comma between a subject and a verb when there is no comma within the subject).

(b) Tests for probably incomplete constituents, based on the chunk types that follow them.

(c) Tests for unexpected arguments, except in some circumstances. For example, "benefactive" indirect objects ("John baked Mary a cake") are dispreferred if they are not in appropriate semantic classes.

Also, a large, complex collection of positive and negative tests, based on syntactic and semantic factors, are used to distinguish among coordinated and appositive readings, and among alternative attachments of appositives.

If the addition or modification of preference tests does not solve a particular problem, then some more basic changes can be made, such as the introduction of new semantic classes. And, in rare cases, new features are added to output network states in order to make properties of non-head constituents encountered along a path available for testing both further along the path and in the development of higher-level constituents. An example is the person and number of syntactic subjects, allowing contexted preference tests for finite verb phrases to check for subject consistency.

5.1 Relationship to "supervised" training

To illustrate the relationship between the above symbolic training method for preference scoring and corpus-based methods, perhaps the easiest way is to compare it to an adaptation (Collins and Roark, 2004) of the perceptron training method to the problem of obtaining a best parse (either directly, or for parse reranking), because the two methods are analogous in a number of ways.

The basic adapted perceptron training assumes a generator function producing parses for inputs. Each such parse is associated with a vector of feature values that express the number of times the feature appears in the input or parse. The features used are those identified by Roark (2001) for a top-down stochastic parser.

The training method obtains a weight vector W (initially 0) for the feature values, by iterating multiple times over pairs $<x_i, y_i>$ where x_i is a training input, and y_i is the correct parse for x_i. For each pair, the best current parse z_i for x_i produced by the generator, with feature value vector $V(z_i)$, is selected based on the current value of $(W \cdot V(z_i))$. Then if $z_i \neq y_i$, W is incremented by $V(y_i)$, and decremented by $V(z_i)$. After training, the weights in W are divided by the number of training steps (# inputs * # iterations).

The method is analogous to the RH manual training process for preference in a number of ways. First, the features used were developed for suitability to a top-down parser, for example taking into account superordinate categories at several levels, some lexical information associated with non-head, left-side siblings of a node, and some right-hand lookahead. Although only one superordinate category is routinely used in RH preference tests, in order to allow caching of output networks for a category, the preference system allows for and occasionally uses the promotion of non-head features of nested constituents to provide similar capability.

Also, the feature weights obtained by the perceptron training method can be seen to focus on patterns that actually matter in distinguishing correct from incorrect parses, as does RH preference training. Intuitively, the difference is that while symbolic training for RH explicitly pinpoints patterns that distinguish among parses, the perceptron training method accomplishes something similar by postulating some more general features as negative or positive based on particular examples, but allowing the iterations over a large training set to filter out potentially indicative patterns that do not actually serve as such.

These analogies highlight the fact that preference system training, whether symbolic or corpus-based, is ultimately an empirical engineering exercise.

6 Some Experimental Results

Tables 2, 3, and 4 summarize some recent results as obtained by testing on Wall Street Journal section 23 of the Penn Treebank (Marcus et al. 1994). The RH results were obtained by about 8 weeks of manual training on the genre.

Table 2 compares speed and coverage for RH and Collins Model3 (Collins, 1999) run on the same CPU. The table also extrapolates the results to two other parsers, based on reported comparisons with Collins. One extrapolation is to a very fast stochastic parser by Sagae and Lavie (2005). The comparison indicates that the RH parser speed is close to that of the best contemporary parsers.

The second extrapolation is to the LFG XLE parser (Kaplan et al. 2004) for English, consisting of a highly developed symbolic parser and grammar, an OT-based preference component, and a stochastic back end to select among remaining alternative parser outputs. Two sets of values are given for XLE, one obtained using the full English grammar, and one obtained using a reduced grammar ignoring less-frequently applicable rules. The extrapolation indicates that the coverage of RH is quite good for a symbolic parser with limited training on an idiom.

While the most important factor in RH parser speed is the enormous speed of the shallow parser, the preference and pruning approach of the overlay parser make contributions to both speed and coverage. This can be seen in Table 2 by the difference between RH parser results with and without pruning. Pruning increases coverage because without it more parses exceed imposed resource limits.

Table 3 compares accuracy. The values for Collins and Sagae/Lavie are based on comparison with treebank data for the entire section 23. However, because RH does not produce treebank-style tags, the RH values are based only on a random

	Time	No full parse
Sagae/Lavie	***~ 4 min***	*1.1%*
RH Prune	5 min 14 sec	10.8%
RH NoPrune	7 min 5 sec	13.9 %
Collins m3	16 min	**.6%**
XLE reduced	*~24 minutes*	unknown
XLE full	*~80 minutes*	~21%

Table 2. Speeds and *Extrapolated speeds*

	Fully accurate	F-score	Avg cross bkts
Sagae/Lavie	unknwn	86%	unknwn
Collins Lbl	33.6%	88.2%	1.05
CollinsNoLbl	35.4%	**89.4 %**	1.05
RH NoLbl	**46%**	86 %	**.59**

Table 3. Accuracy Comparison

	Average	Median
RH Base	137.10	11
RH Pref	**5.04**	**2**

Table 4. Highest Scoring Parses per Input

100-sentence sample from section 23, and compared using a different unlabeled bracketing standard. For details see Newman (2007). For non-parsed sentences the chunks are bracketed. Accuracy is not extrapolated to XLE because available measurements give f-scores (all $\leq$ 80%) for dependency relations rather than for bracketed constituents.

As a partial indication of the role and effectiveness of the RH preference system, if non-parsed sentences are ignored, the percentage of fully accurate bracketings shown in Table 3 rises to approximately 46/89 = 51.6% (it is actually larger because coverage is higher on the 100-sentence sample). As further indication, Table 4 compares, for section 23, the average and median number of parses per sentence obtained by the base grammar alone (RH Base), and the base grammar plus the preference system (RH Pref).[3] The table demonstrates that the preference system is a crucial parser component. Also, the median of 2 parses per sentence obtained using the preference system explains why the fallback low-attach strategy is successful in many cases.

7 Summary and Directions

The primary contribution of this work is in demonstrating the feasibility of a vastly simplified symbolic preference scoring method. The preference scores assigned are neither "principle-based", nor "ad-hoc", but explicitly engineered to facilitate the management of undesirable interactions in the grammar and in the preference system itself. Restricting individual contexted scores to {-1, 0, +1} addresses the problems of multi-level contexted scoring discussed in Section 2, as follows:

- No abstract judgment is required to assign a value to a preference or dispreference.
- Information deficiencies contribute only small dispreferences, so they can often be overcome by preferences.
- Compensating for interactions that are foreseen does not require searching the rules to find necessary compensating values.
- For unforeseen interactions discovered when reviewing parser results, the simplified preference scores facilitate finding the sources of the problems and potential methods of solving them.

[3] The values are somewhat inflated because they include duplicate parses, which have not yet been entirely eliminated.

This approach to symbolic preference has facilitated development and maintenance of the RH parser, and has enabled the production of results with a speed and accuracy comparable to the best stochastic parsers, even with limited training on an idiom.

An interesting question is why this very simple approach does not seem to have been used previously. Part of the answer may lie in the lack of explicit recognition that symbolic preference scoring is ultimately an engineering problem, and is only indirectly based on cognitive principles or approximations to frequencies of particular relationships.

Ongoing development of the RH preference system includes continuing refinement based on "manual" training, and continuing expansion of the set of semantic features used as the parser is applied to new domains. Additional development will also include more encoding of, and attention to, the expected semantic features of arguments. Experiments are also planned to examine the accuracy/performance tradeoffs of using additional context information in the preference tests.

References

Salah Aït-Mokhtar, Jean-Pierre Chanod, Claude Roux. 2002. Robustness beyond shallowness: incremental deep parsing, *Natural Language Engineering* 8:121-144, Cambridge University Press.

Aoife Cahill, Michael Burke, Ruth O'Donovan, Josef van Genabith, and Andy Way. 2004. Long-Distance Dependency Resolution in Automatically Acquired Wide-Coverage PCFG-Based LFG Approximations, In *Proc of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL'04)*, Barcelona

Michael Collins. 1999. Head-Driven Statistical Models for Natural Language Parsing. Ph.D. thesis, University of Pennsylvania.

Michael Collins and Brian Roark. 2004. Incremental Parsing with the Perceptron Algorithm. In *Proc of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL'04)*, Barcelona.

Martin Forst, Jonas Kuhn, and Christian Rohrer. 2005. Corpus-based Learning of OT Constraint Rankings for Large-scale LFG Grammars. In *Proc of the*

LFG05 Conference, Bergen. Available at http://cslipublications.stanford.edu/LFG/10/lfg05.pdf

Anette Frank., Tracy H. King, Jonas Kuhn, and John Maxwell. 1998. Optimality theory style constraint ranking in large-scale LFG grammars. In M. Butt and T. H. King, Eds. *Proc of the Third LFG Conference.* Available http://csli-publications.stanford.edu/LFG3/ Revised version in Peter Sells, ed. *Formal and Theoretical Issues in Optimality Theoretic Syntax*, CSLI Publications, 2001.

Nuria Gala. 2004. Using a robust parser grammar to automatically generate UNL graphs. *In Proc Workshop on Robust Methods for Natural Language Data at COLING'04,* Geneva

Donald Hindle and Mats Rooth. 1993. Structural Ambiguity and Lexical Relations. *Computational Linguistics*, 19:1,103–120.

Jerry R. Hobbs and John Bear. 1990. Two Principles of Parse Preference. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING'90),* Helsinki, Finland, August 1990.

Jerry. R. Hobbs, Douglas E. Appelt, John Bear, Mabry Tyson, and David Magerman. 1992. Robust processing of real-world natural language texts. In Paul S. Jacobs, ed., *Text-Based Intelligent Systems: Current Research and Practice in Information Extraction and Retrieval.* Lawrence Erlbaum, New Jersey, 1992.

Mark Johnson. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL'02),* Philadelphia, July 2002, pp. 136-143.

Ronald M. Kaplan, Stephan Riezler, Tracy H. King, John T. Maxwell, Alex Vasserman. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proc HLT/NAACL'04,* Boston, MA.

Chao-Lin Liu, Jing-Shin Chang, and Keh-Yi Su. 1990. The Semantic Score Approach to the Disambiguation of PP Attachment Problem. In *Proceedings of ROCLING-90*, Taiwan

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a large annotated corpus of English: The Penn treebank**.** *Computational Linguistics***,** 19(2) pp.313--330**.**

Michael C. McCord. 1993. Heuristics for Broad-Coverage Natural Language Parsing. *Proceedings of the workshop on Human Language Technology 1993*, Princeton, NJ

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. Non-projective dependency parsing using spanning tree algorithms**.** In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing* (*EMNLP 2005)*, Vancouver.

Paula S. Newman. 2007. RH: A Retro-Hybrid Parser. In *Short Papers of Proceedings of NAACL/HLT 2007*, Rochester NY

Alan Prince and Paul Smolensky (1993). Optimality Theory: Constraint interaction in generative grammar, Rutgers University Center for Cognitive Science, New Brunswick, NJ. Report RuCCS-TR-2. [Reprinted in John J McCarthy, ed., *Optimality Theory in Phonology: A Book of Readings*, Blackwell (2003).]

Philip Resnik and Marti Hearst. 1993. *Structural Ambiguity and Conceptual Relations,* in Proc. of 1st Workshop on Very Large Corpora, 1993.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. In *Computational Linguistics*, 27(2), pages 249-276.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proc. 9th Int'l Workshop on Parsing Technologies*. Vancouver

Kristina Toutanova, Christopher D. Manning, Dan Flickinger, and Stephan Oepen. 2005. Stochastic HPSG Parse Disambiguation using the Redwoods Corpus. *Research in Language and Computation* 2005.

Yorick A. Wilks. 1975**.** An Intelligent Analyzer and Understander of English**.** *Communications of the ACM* 18(5), pp.264-274

William Woods. 1970. Transition network grammars for natural language analysis. *Communications of the ACM* 13(10), pp.591-606

XLE Online Documentation. 2006. Available at http://www2.parc.com/isl/groups/nltt/xle/doc/xle.html#SEC15

Kiyoshi Yamabana, Shin'ichiro Kamei and Kazunori Muraki. On Representation of Preference Scores. In Proceedings of *The Fifth International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-93),* Kyoto, pp. 92-101

Synchronous Grammars and Transducers: Good News and Bad News

Stuart M. Shieber
School of Engineering and Applied Sciences
Harvard University
Cambridge MA 02138
USA
shieber@seas.harvard.edu

Much of the activity in linguistics, especially computational linguistics, can be thought of as characterizing not languages simpliciter but relations among languages. Formal systems for characterizing language relations have a long history with two primary branches, based respectively on tree transducers and synchronous grammars. Both have seen increasing use in recent work, especially in machine translation. Indeed, evidence from millennia of experience with bilingual dictionaries argues for synchronous grammars as an appropriate substrate for statistical machine translation systems.

On the positive side, some new results have integrated the two branches through the formal-language-theoretic construct of the bimorphism. I will present some background on this integration, and briefly describe two applications of synchronous grammars: to tree-adjoining grammar semantics and to syntax-aware statistical machine translation.

On the negative side, algorithms for making use of these formalisms are computationally complex, perhaps prohibitively so. I will close with a plea for novel research by the parsing technology community in making the systems practical.

Are Very Large Context-Free Grammars Tractable?

Pierre Boullier & Benoît Sagot
INRIA-Rocquencourt
Domaine de Voluceau, Rocquencourt BP 105
78153 Le Chesnay Cedex, France
{Pierre.Boullier,Benoit.Sagot}@inria.fr

Abstract

In this paper, we present a method which, in practice, allows to use parsers for languages defined by very large context-free grammars (over a million symbol occurrences). The idea is to split the parsing process in two passes. A first pass computes a sub-grammar which is a specialized part of the large grammar selected by the input text and various filtering strategies. The second pass is a traditional parser which works with the sub-grammar and the input text. This approach is validated by practical experiments performed on a Earley-like parser running on a test set with two large context-free grammars.

1 Introduction

More and more often, in real-word natural language processing (NLP) applications based upon grammars, these grammars are no more written by hand but are automatically generated, this has several consequences. This paper will consider one of these consequences: the generated grammars may be very large. Indeed, we aim to deal with grammars that have, say, over a million symbol occurrences and several hundred thousands rules. Traditional parsers are not usually prepared to handle them, either because these grammars are simply too big (the parser's internal structures blow up) or the time spent to analyze a sentence becomes prohibitive.

This paper will concentrate on context-free grammars (CFG) and their associated parsers. However, virtually all Tree Adjoining Grammars (TAG, see e.g., (Schabes et al., 1988)) used in NLP applications can (almost) be seen as lexicalized Tree Insertion Grammars (TIG), which can be converted into strongly equivalent CFGs (Schabes and Waters, 1995). Hence, the parsing techniques and tools described here can be applied to most TAGs used for NLP, with, in the worst case, a light over-generation which can be easily and efficiently eliminated in a complementary pass. This is indeed what we have achieved with a TAG automatically extracted from (Villemonte de La Clergerie, 2005)'s large-coverage factorized French TAG, as we will see in Section 4. Even (some kinds of) non CFGs may benefit from the ideas described in this paper.

The reason why the run-time of context-free (CF) parsers for large CFGs is damaged relies on a theoretical result. A well-known result is that CF parsers may reach a worst-case running time of $\mathcal{O}(|G| \times n^3)$ where $|G|$ is the *size* of the CFG and n is the *length* of the source text.[1] In typical NLP applications which mainly work at the sentence level, the length of a sentence does not often go beyond a value of say 100, while its average length is around 20-30 words.[2] In these conditions, the size of the grammar, despite its linear impact on the complexity, may be the prevailing factor: in (Joshi, 1997), the author remarks that "the real limiting factor in practice is the size of the grammar".

The idea developed in this paper is to split the parsing process in two passes. A first pass called *filtering* pass computes a sub-grammar which is the

[1] These two notions will be defined precisely later on.
[2] At least for French, English and similar languages.

sub-part of the large input grammar selected by the input sentence and various filtering strategies. The second pass is a traditional parser which works with the sub-grammar and the input sentence. The purpose is to find a filtering strategy which, in typical practical situations, minimizes on the average the total run-time of the filtering pass followed by the parser pass.

A filtering pass may be seen as a (filtering) function that uses the input sentence to select a sub-grammar out of a large input CFG. Our hope, using such a filter, is that the time saved by the parser pass which uses a (smaller) sub-grammar will not totally be used by the filter pass to generate this sub-grammar.

It must be clear that this method cannot improve the worst-case parse-time because there exists grammars for which the sub-grammar selected by the filtering pass is the input grammar itself. In such a case, the filtering pass is simply a waste of time. Our purpose in this paper is to argue that this technique may profit from typical grammars used in NLP. To do that we put aside the theoretical view point and we will consider instead the average behaviour of our processors.

More precisely we will study on two large NL CFGs the behaviour of our filtering strategies on a set of test sentences. The purpose being to choose the *best* filtering strategy, if any. By best, we mean the one which, on the average, minimizes the total run-time of both the filtering pass followed by the parsing pass.

Useful formal notions and notations are recalled in Section 2. The filtering strategies are presented in Section 3 while the associated experiments are reported in Section 4. This paper ends with some concluding remarks in Section 5.

2 Preliminaries

2.1 Context-free grammars

A CFG G is a quadruple (N, T, P, S) where N is a non-empty finite set of *nonterminal symbols*, T is a finite set of *terminal symbols*, P is a finite set of (context-free rewriting) *rules* (or *productions*) and S is a distinguished nonterminal symbol called the *axiom*. The sets N and T are disjoint and $V = N \cup T$ is the *vocabulary*. The rules in P have the form $A \rightarrow \alpha$, with $A \in N$ and $\alpha \in V^*$.

For a given string $\alpha \in V^*$, its size (length) is noted $|\alpha|$. As an example, for the input string $w = a_1 \cdots a_n$, $a_i \in T$, we have $|w| = n$. The empty string is denoted ε and we have $|\varepsilon| = 0$. The size $|G|$ of a CFG G is defined by $|G| = \sum_{A \rightarrow \alpha \in P} |A\alpha|$.

For G, on strings of V^*, we define the binary relation *derive*, noted $\Rightarrow$, by $\gamma_1 A \gamma_2 \overset{A\rightarrow\alpha}{\underset{G}{\Rightarrow}} \gamma_1 \alpha \gamma_2$ if $A \rightarrow \alpha \in P$ and $\gamma_1, \gamma_2 \in V^*$. The subscript G or even the superscript $A \rightarrow \alpha$ may be omitted. As usual, its transitive (resp. reflexive transitive) closure is noted $\overset{+}{\underset{G}{\Rightarrow}}$ (resp. $\overset{*}{\underset{G}{\Rightarrow}}$). We call *derivation* any sequence of the form $\gamma_1 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} \gamma_2$. A *complete derivation* is a derivation which starts with the axiom and ends with a terminal string w. In that case we have $S \overset{*}{\underset{G}{\Rightarrow}} \gamma \overset{*}{\underset{G}{\Rightarrow}} w$, and γ is a *sentential form*.

The *string language* defined (generated, recognized) by G is the set of all the terminal strings that are derived from the axiom: $\mathcal{L}(G) = \{w \mid S \overset{+}{\underset{G}{\Rightarrow}} w, w \in T^*\}$. We say that a CFG is empty iff its language is empty.

A nonterminal symbol A is *nullable* iff it can derive the empty string (i.e., $A \overset{+}{\underset{G}{\Rightarrow}} \varepsilon$). A CFG is ε*-free* iff its nonterminal symbols are non-nullable.

A CFG is *reduced* iff every symbol of every production is a symbol of at least one complete derivation. A reduced grammar is empty iff its production set is empty ($P = \emptyset$). We say that a non-empty reduced grammar is in *canonical form* iff its vocabulary only contains symbols that appear in the productions of P.[3,4]

Two CFGs G and G' are *weakly equivalent* iff they generate the same string language. They are *strongly equivalent* iff they generate the same set of structural descriptions (i.e., parse trees). It is a well known result (See Section 3.2) that every CFG G can be transformed in time linear w.r.t. $|G|$ into a strongly equivalent (canonical) reduced CFG G'.

For a given input string $w \in T^*$, we define its

[3]We may say that the canonical form of the empty reduced grammar is $(\{S\}, \emptyset, \emptyset, S)$ though the axiom S does not appear in any production.

[4]Note that the pair (P, S) completely defines a reduced CFG $G = (N, T, P, S)$ in canonical form since we have $N = \{X_0 \mid X_0 \rightarrow \alpha \in P\} \cup \{S\}$, $T = \{X_i \mid X_0 \rightarrow X_1 \cdots X_p \in P \wedge 1 \leq i \leq p\} - N$. Thus, in the sequel, we often note simply $G = (P, S)$ grammars in canonical form.

ranges as the set $R^w = \{[i..j] \mid 1 \leq i \leq j \leq |w| + 1\}$. If $w = w_1 t w_3 \in T^*$ is a terminal string, and if $t \in T \cup \{\varepsilon\}$ is a (terminal or empty) symbol, the *instantiation* of t in w is the triple noted $t[i..j]$ where $[i..j]$ is a range with $i = |w_1| + 1$ and $j = i + |t|$. More generally, the *instantiation* of the terminal string w_2 in $w_1 w_2 w_3$ is noted $w_2[i..j]$ with $i = |w_1| + 1$ and $j = i + |w_2|$. Obviously, the instantiation of w itself is then $w[1..1 + |w|]$.

Let us consider an input string $w = w_1 w_2 w_3$ and a CFG G. If we have a complete derivation $d = S \overset{*}{\underset{G}{\Rightarrow}} w_1 A w_3 \overset{A \rightarrow \alpha}{\underset{G}{\Rightarrow}} w_1 \alpha w_3 \overset{*}{\underset{G}{\Rightarrow}} w_1 w_2 w_3$, we see that A derives w_2 (we have $A \overset{+}{\underset{G}{\Rightarrow}} w_2$). Moreover, in this complete derivation, we also know a range in R^w, namely $[i..j]$, which covers the substring w_2 which is derived by A ($i = |w_1| + 1$ and $j = i + |w_2|$). This is represented by the *instantiated nonterminal symbol* $A[i..j]$. In fact, each symbol which appears in a complete derivation may be transformed into its instantiated counterpart. We thus talk of instantiated productions or (complete) instantiated derivations. For a given input text w, and a CFG G, let P_G^w be the set of instantiated productions that appears in all complete instantiated derivations.[5] The pair $(P_G^w, S[1..|w| + 1])$ is the *(reduced) shared parse forest* in canonical form.[6]

2.2 Finite-state automata

A *finite-state automaton* (FSA) is the 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$ where Q is a non empty finite set of *states*, Σ is a finite set of *terminal symbols*, δ is the transition relation $\delta = \{(q_i, t, q_j) | q_i, q_j \in Q \wedge t \in T \cup \{\varepsilon\}\}$, q_0 is a distinguished element of Q called the *initial state* and F is a subset of Q whose elements are called *final states*. The size of A is defined by $|A| = |\delta|$.

As usual, we define both a *configuration* as an element of $Q \times T^*$ and *derive* a binary relation between configurations, noted $\underset{A}{\vdash}$ by $(q, tx) \underset{A}{\vdash} (q', x)$, iff $(q, t, q') \in \delta$. If $w'w'' \in T^*$, we call *derivation* any sequence of the form $(q', w'w'') \underset{A}{\vdash} \cdots \underset{A}{\vdash} (q'', w'')$. If $w \in T^*$, the *initial configuration* is noted c_0 and is the pair (q_0, w). A *final configuration* is noted c_f and has the form (q_f, ε) with $q_f \in F$. A *complete derivation* is a derivation which starts with c_0 and ends in a final configuration c_f. In that case we have $c_0 \overset{*}{\underset{A}{\vdash}} c_f$.

The *language* $\mathcal{L}(A)$ *defined* (*generated*, *recognized*) by the FSA A is the set of all terminal strings w for which there exists a complete derivation. We say that an FSA is empty iff its language is empty. Two FSAs A and A' are *equivalent* iff they defined the same language.

An FSA is *ε-free* iff its transition relation has the form $\delta = \{(q_i, t, q_j) | q_i, q_j \in Q, t \in \Sigma\}$, except perhaps for a distinguished transition, the *ε-transition* which has the form (q_0, ε, q_f), $q_f \in F$ and allows the empty string ε to be in $\mathcal{L}(A)$. Every FSA can be transformed into an equivalent ε-free FSA.

An FSA $A = (Q, \Sigma, \delta, q_0, F)$ is *reduced* iff every element of δ appears in a complete derivation. A reduced FSA is empty iff we have $\delta = \emptyset$. We say that a non-empty reduced FSA is in *canonical form* iff its set of states Q and its set of terminal symbols Σ only contain elements that appear in the transition relation δ.[7] It is a well known result that every FSA A can be transformed in time linear with $|A|$ into an equivalent (canonical) reduced FSA A'.

2.3 Input strings and input DAGs

In many NLP applications[8] the source text cannot be considered as a single string of terminal symbols but rather as a finite set of terminal strings. These sets are finite languages which can be defined by particular FSAs. These particular type of FSAs are called *directed-acyclic graphs* (DAGs). In a DAG $w = (Q, \Sigma, \delta, q_0, F)$, the initial state q_0 is 1 and we assume that there is a single final state f ($F = \{f\}$), Q is a finite subset of the positive integers less than or equal to f: $Q = \{i | 1 \leq i \leq f\}$, Σ is the set of terminal symbols. For the transition relation δ, we

[5]For example, in the previous complete derivation d, let the right-hand side α be the (vocabulary) string $X_1 \cdots X_k \cdots X_p$ in which each symbol X_k derives the terminal string $x_k \in T^*$ (we have $X_k \overset{*}{\underset{G}{\Rightarrow}} x_k$ and $w_2 = x_1 \cdots x_k \cdots x_p$), then the instantiated production $A[i_0..i_p] \rightarrow X_1[i_0..i_1] \cdots X_k[i_{k-1}..i_k] \cdots X_p[i_{p-1}..i_p]$ with $i_0 = |w_1| + 1$, $i_1 = i_0 + |x_1|, \ldots, i_k = i_{k-1} + |x_k| \ldots$ and $i_p = i_0 + |w_2|$ is an element of P_G^w.

[6]The popular notion of shared forests mainly comes from (Billot and Lang, 1989).

[7]We may say that the canonical form of the empty reduced FSA is $(\{q_0\}, \emptyset, \emptyset, q_0, \emptyset)$ though the initial state q_0 does not appear in any transition.

[8]Speech processing, lexical ambiguity representation, …

require that its elements (i, t, j) are such that $i < j$ (there are no loops in a DAG). Without loss of generality, we will assume that DAGs are ε-free reduced FSAs in canonical form and that any DAG w is noted by a triple (Σ, δ, f) since its initial state is always 1 and its set of states is $\{i \mid 1 \leq i \leq f\}$.

For a given CFG G, the recognition of an input DAG w is equivalent to the emptiness of its intersection with G. This problem can be solved in time linear in $|G|$ and cubic in $|Q|$ the number of states of w.

If the input text is a DAG, the previous notions of range, instantiations and parse forest easily generalize: the indices i and j which in the string case locate the positions of substrings are changed in the DAG case into DAG states. For example if $A[i_0..i_p] \rightarrow X_1[i_0..i_1] \cdots X_p[i_{p-1}..i_p]$ is an instantiated production of the parse forest for $G = (N, T, P, S)$ and $w = (\Sigma, \delta, f)$, we have $A \rightarrow X_1 \cdots X_p \in P$ and there is a path in the input DAG from state i_0 to state i_p via states $i_1, \ldots, i_{p-1}$.

Of course, any nonempty terminal string $w \in T^+$, may be viewed as a DAG (Σ, δ, f) where $\Sigma = \{t \mid w = w_1 t w_2 \wedge t \in T\}$, $\delta = \{(i, t, i+1) \mid w = w_1 t w_2 \wedge t \in T \wedge i = 1 + |w_1|\}$ and $f = 1 + |w|$. If the input string w is the empty string ε, the associated DAG is (Σ, δ, f) where $\Sigma = \emptyset$, $\delta = \{(1, \varepsilon, 2)\}$ and $f = 2$. Thus, in the sequel, we will assume that the inputs of our parsers are not strings but DAGs. As a consequence the size (or *length*) of a sentence is the size of its DAG (i.e., its number of transitions).

3 Filtering Strategies

3.1 Gold Strategy

Let $G = (N, T, P, S)$ be a CFG, $w = (\Sigma, \delta, f)$ be an input DAG of size $n = |\delta|$ and $\langle F_w \rangle = (\langle P_w \rangle, S[1..f])$ be the reduced output parse forest in canonical form. From $\langle P_w \rangle$, it is possible to extract a set of (reduced) uninstantiated productions $P_w^g = \{A \rightarrow X_1 \cdots X_p \mid A[i_0..i_p] \rightarrow X_1[i_0..i_1] X_2[i_1..i_2] \cdots X_p[i_{p-1}..i_p] \in \langle P_w \rangle\}$, which, together with the axiom S, defines a new reduced CFG $G_w^g = (P_w^g, S)$ in canonical form. This grammar is called the *gold* grammar of G for w, hence the superscript g. Now, if we use G_w^g to reparse the same input DAG w, we will get the same output forest $\langle F_w \rangle$. But in that case, we are sure that every production in P_w^g is used in at least one complete derivation. Now, if this process is viewed as a filtering strategy that computes a filtering function as introduced in Section 1, it is clear that this strategy is *size-optimal* in the sense that P_w^g is of minimal size, we call it the *gold* strategy and the associated gold filtering function is noted g. Since we do not want that a filtering strategy looses parses, the result $G_w^f = (P_w^f, S)$ of any filtering function f must be such that, for every sentence w, P_w^f is a superset of P_w^g. In other words the *recall score* of any filtering function f must be of 100%. We can note that the parsing pass which generates G_w^g may be led by any filtering strategy f.

As usual, the *precision score* (precision for short) of a filtering strategy f (w.r.t. the gold case) is, for a given w, defined by the quotient $\frac{|P_w^g|}{|P_w^f|}$ which expresses the number of useful productions selected by f on w (for some G).

However, it is clear that we are interested in strategies that are *time-optimal* and size-optimal strategies are not necessarily also time-optimal: the time taken at filtering-time to get a smaller grammar will not necessarily be won back at parse-time.

For a given CFG G, an input DAG w and a filtering strategy c, we only have to plot the times taken by the filtering pass and by the parsing pass to make some estimations on their average (median, decile) parse times and then to decide which is the winner. However, it may well happens that a strategy which has not received the award (with the sample of CFGs and the test sets tried) would be the winner in another context!

All the following filtering strategies exhibit necessary conditions that any production must hold in order to be in a parse.

3.2 The *make-a-reduced-grammar* Algorithm

An algorithm which takes as input any CFG $G = (N, T, P, S)$ and generates as output a strongly equivalent *reduced* CFG G' and which runs in $\mathcal{O}(|G|)$ can be found in many text books (See (Hopcroft and Ullman, 1979) for example).

So as to eliminate from all our intermediate subgrammars all useless productions, each filtering strategy will end by a call to such an algorithm named *make-a-reduced-grammar*.

The *make-a-reduced-grammar* algorithm works as follows. It first finds all *productive*[9] symbols. Afterwards it finds all *reachable*[10] symbols. A symbol is *useful* (otherwise *useless*) if it is both productive and reachable. A production $A \rightarrow X_1 \cdots X_p$ is *useful* (otherwise *useless*) iff all its symbols are useful. A last scan over the grammar erases all useless production and leaves the reduced form. The *canonical form* is reached in only retaining in the nonterminal and terminal sets of the sub-grammar the symbols which occur in the (useful) production set.

3.3 Basic Filtering Strategy: b-filter

The basic filtering strategy (b-filter for short) which is described in this section will always be tried the first. Thus, its input is the couple (G, w) where $G = (N, T, P, S)$ is the large initial CFG and the input sentence w is a reduced DAG in canonical form $w = (\Sigma, \delta, f)$ of size n. It generates a reduced CFG in canonical form noted $G^b = (P^b, S)$ in which the references to both G and w are assumed. Besides this b-filter, we will examine in Sections 3.4 and 3.5 two others filtering strategies named a and d. These filters will always have as input a couple (G^c, w) where $G^c = (P^c, S)$ is a reduced CFG in canonical form which has already been filtered by a previous sequence of strategies noted c. They generate a reduced CFG in canonical form noted $G^{cf} = (P^{cf}, S)$ with $f = a$ or $f = d$ respectively. Of course it may happens that G^{cf} is identical to G^c if the f-filter is not effective. A filtering strategy or a combination of filtering strategies may be applied several times and lead to a filtered grammar of the form say G^{ba^2da} in which the sequence ba^2da explicits the order in which the filtering strategies have been performed. We may even repeatedly apply a until a fixed point is reached before applying d, and thus get something of the form $G^{ba^\infty d}$.

The idea behind the b-filter is very simple and has largely been used in lexicalized formalisms parsing, in particular in LTAG (Schabes et al., 1988) parsing. The filter rejects productions of P which contain terminal symbols that do not occur in Σ (i.e., that are not terminal symbols of the DAG w) and thus takes $\mathcal{O}(|G|)$ time if we assume that the access to the elements of the terminal set Σ is performed in constant time. *Unlexicalized* productions whose right-hand sides are in N^* are kept. It also rejects productions in which several terminal symbol occurs, in an order which is not compatible with the linear order of the input.

Consider for example the set of productions shown in Table 1 and assume that the source text is the terminal string ab. It is clear that the b-filter will erase production 6 since c is not in the source text.

$$\begin{aligned} S &\rightarrow AB & (1)\\ S &\rightarrow BA & (2)\\ A &\rightarrow a & (3)\\ A &\rightarrow ab & (4)\\ B &\rightarrow b & (5)\\ B &\rightarrow bc & (6) \end{aligned}$$

Table 1: A simple grammar

The execution of the b-filter produces a (non-reduced) CFG G' such that $|G'| \leq |G|$. However, it may be the case that some productions of G' are useless, it will thus be the task of the *make-a-reduced-grammar* algorithm to transform G' into its reduced canonical form G^b in time $\mathcal{O}(|G'|)$. The worst-case total running time of the whole b-filter pass is thus $\mathcal{O}(|G| \times n)$.

We can remark that, after the execution of the b-filter, the set of terminal symbols of G^b is a subset of $T \cap \Sigma$.

3.4 Adjacent Filtering Strategy: a-filter

As explained before, we assume that the input to the adjacent filtering strategy (a-filter for short) described in this section is a couple (G^c, w) where $G^c = (N^c, T^c, P^c, S)$ is a reduced CFG in canonical form. However, the a-filter would also work for a non-reduced CFG. As usual, we define the symbols of G^c as the elements of the vocabulary $V^c = N^c \cup T^c$.

The idea is to erase productions that cannot be part of any parses for w in using an adjacency criteria: if two symbols are adjacent in a rule, they must

[9] $X \in V$ is productive iff we have $X \overset{*}{\underset{G}{\Rightarrow}} w, w \in T^*$.

[10] $X \in V$ is reachable iff we have $S \overset{*}{\underset{G}{\Rightarrow}} w_1 X w_2, w_1 w_2 \in T^*$.

derive terminal symbols that are also adjacent in w. To give a (very) simple practical idea of what we mean by adjacency criteria, let us consider again the source string ab and the grammar defined in Table 1 in which the last production has already been erased by the b-filter.

The fact that the B-production ends with a b and that the A-productions all start with an a, implies that production 2 is in a complete parse only if the source text is such that b is immediately followed by a. Since it is not the case, production 2 can be erased.

More generally, consider a production of the form $A \rightarrow \cdots XY \cdots$. If for each couple $(a, b) \in T^2$ in which a is a terminal symbol that can terminate (the terminal strings generated by) X and b is a terminal symbol that can lead (the terminal strings generated by) Y, there is no transition on b that can follow a transition on a in the DAG w, it is clear that the production $A \rightarrow \cdots XY \cdots$ can be safely erased.

Now assume that we have the following (left) derivation $Y \overset{*}{\Rightarrow} Y_1\beta_1 \overset{*}{\Rightarrow} Y_i\beta_i \cdots \beta_1 \overset{*}{\Rightarrow} \ldots \overset{Y_{p-1} \rightarrow \alpha_p Y_p \beta_p}{\Rightarrow} \alpha_p Y_p \beta_p \cdots \beta_1 \overset{*}{\Rightarrow} Y_p\beta_p \cdots \beta_1$, with $\alpha_p \overset{*}{\Rightarrow} \varepsilon$. If for each couple (a, b') in which a has the previous definition and b' is a terminal symbol that can lead (the terminal strings generated by) Y_p, there is no transition on b' that can follow a transition on a in the DAG w, the production $Y_{p-1} \rightarrow \alpha_p Y_p \beta_p$ can be erased if it is not valid in another context.

Moreover, consider a (right) derivation of the form $X \overset{*}{\Rightarrow} \alpha_1 X_1 \overset{*}{\Rightarrow} \alpha_1 \cdots \alpha_i X_i \overset{*}{\Rightarrow} \ldots \overset{X_{p-1} \rightarrow \alpha_p X_p \beta_p}{\Rightarrow} \alpha_1 \cdots \alpha_p X_p \beta_p \overset{*}{\Rightarrow} \alpha_1 \cdots \alpha_p X_p$, with $\beta_p \overset{*}{\Rightarrow} \varepsilon$. If for each couple (a', b) in which b has the previous definition and a' is a terminal symbol that can terminate (the terminal strings generated by) X_p, there is no transition on b that can follow a transition on a' in the DAG w, the production $X_{p-1} \rightarrow \alpha_p X_p \beta_p$ can be erased if it is not valid in another context.

In order to formalize these notions we define several binary relations together with their (reflexive) transitive closure.

Within a CFG $G = (N, T, P, S)$, we first define *left-corner* noted $\llcorner$. Left-corner (Nederhof, 1993; Moore, 2000), hereafter *LC*, is a well-known relation since many parsing strategies are based upon it. We say that X is in the LC of A and we write $A \llcorner X$ iff $(A, X) \in \{(B, Y) \mid B \rightarrow \alpha Y \beta \in P \wedge \alpha \underset{G}{\overset{*}{\Rightarrow}} \varepsilon\}$. We can write $A \underset{A \rightarrow \alpha X \beta}{\llcorner} X$ to enforce how the couple (A, X) may be produced.

For its dual relation, *right-corner*, noted $\lrcorner$, we say that X is in the right corner of A and we write $X \lrcorner A$ iff $(X, A) \in \{(Y, B) \mid B \rightarrow \alpha Y \beta \in P \wedge \beta \underset{G}{\overset{*}{\Rightarrow}} \varepsilon\}$. We can write $X \underset{A \rightarrow \alpha X \beta}{\lrcorner} A$ to enforce how the couple (X, A) may be produced.

We also define the *first* (resp. *last*) relation noted $\hookrightarrow_t$ (resp. $\hookleftarrow_t$) by $\hookrightarrow_t = \{(X, t) \mid X \in V \wedge t \in T \wedge X \underset{G}{\overset{*}{\Rightarrow}} tx \wedge x \in T^*\}$ (resp. $\hookleftarrow_t = \{(X, t) \mid X \in V \wedge t \in T \wedge X \underset{G}{\overset{*}{\Rightarrow}} xt \wedge x \in T^*\}$).

We define the *adjacent* ternary relation on $V \times N^* \times V$ noted $\leftrightarrow$ and we write $X \overset{\sigma}{\leftrightarrow} Y$ iff $(X, \sigma, Y) \in \{(U, \beta, V) \mid A \rightarrow \alpha U \beta V \gamma \in P \wedge \beta \underset{G}{\overset{*}{\Rightarrow}} \varepsilon\}$. This means that X and Y occur in that order in the right-hand side of some production and are separated by a nullable string σ. Note that X or Y may or may not be nullable.

On the input DAG $w = (\Sigma, \delta, f)$, we define the *immediately precede* relation noted $<$ and we write $a < b$ for $a, b \in \Sigma$ iff $w_1 ab w_3 \in \mathcal{L}(w), w_1, w_3 \in \Sigma^*$.

We also define the *precede* relation noted $\ll$ and we write $a \ll b$ for $a, b \in \Sigma$ iff $w_1 a w_2 b w_3 \in \mathcal{L}(w), w_1, w_2, w_3 \in \Sigma^*$.We can note that $\ll$ is not the transitive closure of $<$.[11]

For each production $A \rightarrow \alpha X_0 X_1 \cdots X_{p-1} X_p \gamma$ in P^c and for each symbol pairs (X_0, X_p) of non-nullable symbols s.t. $X_1 \cdots X_{p-1} \underset{G^c}{\overset{*}{\Rightarrow}} \varepsilon$, we compute two sets A_1 and A_2 of couples $(a, b), a, b \in T^c$ defined by $A_1 = \cup_{0 < i \leq p} = \{(a, b) \mid a \hookleftarrow_t X_0 \overset{X_1 \cdots X_{i-1}}{\leftrightarrow} X_i \hookrightarrow_t b\}$ and $A_2 = \cup_{0 \leq i < p} = \{(a, b) \mid a \hookleftarrow_t X_i \overset{X_{i+1} \cdots X_{p-1}}{\leftrightarrow} X_p \hookrightarrow_t b\}$. Any

[11]Consider the source string $bcab$ for which we have $a \overset{+}{<} c$, but not $a \ll c$.

pair (a, b) of A_1 is such that the terminal symbol a may terminate a phrase of X_0 while the terminal symbol b may lead a phrase of $X_1 \cdots X_p$. Since X_0 and X_p are not nullable, A_1 is not empty. If none of its elements (a, b) is such that $a < b$, the production $A \rightarrow \alpha X_0 X_1 \cdots X_{p-1} X_p \gamma$ is useless and can be erased. Analogously, any pair (a, b) of A_2 is such that the terminal symbol a may terminate a phrase of $X_0 X_1 \cdots X_{p-1}$ while the terminal symbol b may lead a phrase of X_p. Since X_0 and X_p are not nullable, A_2 is not empty. If none of its elements (a, b) is such that $a < b$, the production $A \rightarrow \alpha X_0 X_1 \cdots X_{p-1} X_p \gamma$ is useless and can be erased. Of course if $X_1 \cdots X_{p-1} = \varepsilon$, we have $A_1 = A_2$.[12]

The previous method has checked some adjacent properties inside the right-hand sides of productions. The following will perform some analogous checks but at the beginning and at the end of the right-hand sides of productions.

Let us go back to Table 1 to illustrate our purpose. Recall that, with source text ab, productions 6 and 2 have already been erased. Consider production 4 whose left-hand side is an A, the terminal string ab that it generates ends by b. If we look for the occurrences of A in the right-hand sides of the (remaining) productions, we only find production 1 which indicates that A is followed by B. Since the phrases of B all start with b (See production 5) and since in the source text b does not immediately follow another b, production 4 can be erased.

In order to check that the input sentence w starts and ends by valid terminal symbols, we augment the adjacent relation with two elements $(\$, \varepsilon, S)$ and $(S, \varepsilon, \$)$ where \$ is a new terminal symbol which is supposed to start and to end every sentence.[13]

Let $Z \rightarrow \alpha U \beta$ be a production in P^c in which U is non-nullable and $\alpha \overset{*}{\underset{G_c}{\Rightarrow}} \varepsilon$. If X is a non-nullable symbol, we compute the set $L = \{(a, b) \mid a \hookleftarrow_t X \overset{\sigma}{\leftrightarrow} Y \overset{*}{\llcorner} Z \underset{Z \rightarrow \alpha U \beta}{\llcorner} U \hookrightarrow_t b\}$. Since G^c is reduced and since $\$ < S$, we are sure that the set $X \overset{\sigma}{\leftrightarrow} Y \overset{*}{\llcorner} Z$ is non-empty, thus L is also non-empty.[14]

We can associate with each couple $(a, b) \in L$ at least one (left) derivation of the form $X \sigma Y \overset{*}{\underset{G^c}{\Rightarrow}} w_0 a w_1 \sigma Y \overset{*}{\underset{G^c}{\Rightarrow}} w_0 a w_1 w_2 Y \overset{*}{\underset{G^c}{\Rightarrow}} w_0 a w_1 w_2 w_3 Z \gamma_2 \overset{Z \rightarrow \alpha U \beta}{\underset{G^c}{\Rightarrow}} w_0 a w_1 w_2 w_3 \alpha U \beta \gamma_2 \overset{*}{\underset{G^c}{\Rightarrow}} w_0 a w_1 w_2 w_3 w_4 U \beta \gamma_2 \overset{*}{\underset{G^c}{\Rightarrow}} w_0 a w_1 w_2 w_3 w_4 w_5 b \gamma_1 \beta \gamma_2$ in which $w_1 w_2 w_3 w_4 w_5 \in T^{c*}$. These derivations contains all possible usages of the production $Z \rightarrow \alpha U \beta$ in a parse. If for every couple $(a, b) \in L$, the statement $a \ll b$ does not hold, we can conclude that the production $Z \rightarrow \alpha U \beta$ is not used in any parse and can thus be deleted.

Analogously, we can check that the order of terminal symbols is compatible with both a production and its right grammatical context.

Let $Z \rightarrow \alpha U \beta$ be a production in P^c in which U is non-nullable and $\beta \overset{*}{\underset{G_c}{\Rightarrow}} \varepsilon$. If Y is a non-nullable symbol, we compute the set $R = \{(a, b) \mid a \hookleftarrow_t U \underset{Z \rightarrow \alpha U \beta}{\lrcorner} Z \overset{*}{\lrcorner} X \overset{\sigma}{\leftrightarrow} Y \hookrightarrow_t b\}$. Since G^c is reduced and since $S < \$$, we are sure that the set $Z \overset{*}{\lrcorner} X \overset{\sigma}{\leftrightarrow} Y$ is non-empty, thus R is also non-empty.[14]

To each couple $(a, b) \in R$ we can associate at least one (right) derivation of the form $X \sigma Y \overset{*}{\underset{G^c}{\Rightarrow}} X \sigma w_1 b w_0 \overset{*}{\underset{G^c}{\Rightarrow}} X w_2 w_1 b w_0 \overset{*}{\underset{G^c}{\Rightarrow}} \gamma_1 Z w_3 w_2 w_1 b w_0 \overset{Z \rightarrow \alpha U \beta}{\underset{G^c}{\Rightarrow}} \gamma_1 \alpha U \beta w_3 w_2 w_1 b w_0 \overset{*}{\underset{G^c}{\Rightarrow}} \gamma_1 \alpha U w_4 w_3 w_2 w_1 b w_0 \overset{*}{\underset{G^c}{\Rightarrow}} \gamma_1 \alpha \gamma_2 a w_5 w_4 w_3 w_2 w_1 b w_0$ in which $w_5 w_4 w_3 w_2 w_1 \in T^{c*}$. These derivations contains all possible usages of the production $Z \rightarrow \alpha U \beta$ in a partial parse. If for every couple $(a, b) \in L$, the statement $a \ll b$ does not hold, we can conclude that the production $Z \rightarrow \alpha U \beta$ is not used in any parse and can thus be deleted.

Now, a call to the *make-a-reduced-grammar* algorithm produces a reduced CFG in canonical form named $G^{ca} = (N^{ca}, T^{ca}, P^{ca}, S)$.

[12]It can be shown that the previous check can be performed on (G^c, w) in worst-case time $\mathcal{O}(|G^c| \times |\Sigma|^3)$ (recall that $|\Sigma| \leq n$). This time reduces to $\mathcal{O}(|G^c| \times |\Sigma|^2)$ if the input sentence is not a DAG but a string.

[13]This is equivalent to assume the existence in the grammar of a *super-production* whose right-hand side has the form \$$S$\$.

[14]This statement does not hold any more if we exclude from P^c the productions that have been previously erased during the current a-filter. In that case, an empty set indicates that the production $Z \rightarrow \alpha U \beta$ can be erased.

3.5 Dynamic Set Automaton Filtering Strategy: d-filter

In (Boullier, 2003) the author has presented a method that takes a CFG G and computes a FSA that defines a regular superset of $\mathcal{L}(G)$. However his method would produce intractable gigantic FSAs. Thus he uses his method to dynamically compute the FSA at parse time on a given source text. Based on experimental results, he shows that his method called *dynamic set automaton* (DSA) is tractable. He uses it to *guide* an Earley parser (See (Earley, 1970)) and shows improvements over the non guided version. The DSA method can directly be used as a filtering strategy since the states of the underlying FSA are in fact sets of *items*. For a CFG $G = (N, T, P, S)$, an item (or dotted production) is an element of $\{[A \rightarrow \alpha.\beta] \mid A \rightarrow \alpha\beta \in P\}$. A *complete* item has the form $[A \rightarrow \gamma.]$, it indicates that the production $A \rightarrow \gamma$ has been, in some sense, recognized. Thus, the complete items of the DSA states gives the set of productions selected by the DSA. This selection can be further refined if we also use the mirror DSA which processes the source text from right to left and if we only select complete items that both belong to the DSA and to its mirror.

Thus, if we assume that the input to the DSA filtering strategy (d-filter) is a couple (G^c, w) where $G^c = (P^c, S)$ is a reduced CFG in canonical form, we will eventually get a set of productions which is a subset of P^c. If it is a strict subset, we then apply the *make-a-reduced-grammar* algorithm which produces a reduced CFG in canonical form named $G^{cd} = (P^{cd}, S)$.

The Section 4 will give measures that may help to compare the practical merits of the a and d-filtering strategies.

4 Experiments

The measures presented in this section have been taken on a 1.7GHz AMD Athlon PC with 1.5 Gb of RAM running Linux. All parsers are written in C and have been compiled with gcc 2.96 with the *O2* optimization flag.

4.1 Grammars and corpus

We have performed experiments with two large grammars described below. The first one is an automatically generated CFG, the other one is the CFG equivalent of a TIG automatically extracted from a factorized TAG.

The first grammar, named $G^{T>N}$, is a variant of the CFG backbone of a large-coverage LFG grammar for French used in the French LFG parser described in (Boullier and Sagot, 2005). In this variant, the set T of terminal symbols is the whole set of French inflected forms present in the Le*fff*, a large-coverage syntactic lexicon for French (Sagot et al., 2006). This leads to as many as 407,863 different terminal symbols and 520,711 lexicalized productions (hence, the average number of categories — which are here non-terminal symbols — for an inflected form is 1.27). Moreover, this CFG entails a non-neglectible amount of syntactic constraints (including over-generating sub-categorization frame checking), which implies as many as $|P_u| = 19,028$ non-lexicalized productions. All in all, $G^{T>N}$ has 539,739 productions.

The second grammar, named G^{TIG}, is a CFG which represents a TIG. To achieve this, we applied (Boullier, 2000)'s algorithm on the unfolded version of (Villemonte de La Clergerie, 2005)'s factorized TAG. The number of productions in G^{TIG} is comparable to that of $G^{T>N}$. However, these two grammars are completely different. First, G^{TIG} has much less terminal and non-terminal symbols than $G^{T>N}$. This means that the basic filter may be less efficient on G^{TIG} than on $G^{T>N}$. Second, the size of G^{TIG} is enormous (more than 10 times that of $G^{T>N}$), which shows that right-hand sides of G^{TIG}'s productions are huge (the average number of right-hand side symbols is more than 24). This may increase the usefulness of a- and d-filtering strategies.

Global quantitative data about these grammars is shown in Table 2.

Both grammars, as evoked in the introduction, have not been written by hand. On the contrary, they are automatically generated from a more abstract and more compact level (a meta-level over LFG for $G^{T>N}$, and a metagrammar for G^{TIG}). These grammars are not artificial grammars set up only for this experiment. On the contrary, they are automatically generated huge real-life CFGs that are variants of grammars used in real NLP applications.

Our test suite is a set of 3093 French journalistic sentences. These sentences are the *general_lemonde*

G	$\|N\|$	$\|T\|$	$\|P\|$	$\|P_u\|$	$\|G\|$
$G^{T>N}$	7,862	407,863	539,739	19,028	1,123,062
G^{TIG}	448	173	493,408	4,338	12,455,767

Table 2: Sizes of the grammars $G^{T>N}$ and G^{TIG} used in our experiments

part of the EASy parsing evaluation campaign corpus. Raw sentences have been turned into DAGs of inflected forms known by both grammar/lexicon couples.[15] This step has been achieved by the pre-syntactic processing chain SxPipe (Sagot and Boullier, 2005). They are all recognized by both grammars.[16] The resulting DAGs have a median size of 28 and an average size of 31.7.

Before entering into details, let us give here the first important result of these experiments: it was actually possible to build parsers out of $G^{T>N}$ and G^{TIG} and to parse efficiently with the resulting parsers (we shall detail later on efficiency results). Given the fact that we are dealing with grammars whose sizes are respectively over 1,000,000 and over 12,000,000, this is in itself a very satisfying result.

4.2 Precision results

Let us recall informally that the precision of a filtering strategy is the proportion of productions in the resulting sub-grammar that are in the gold grammar, i.e., that have effectively instantiated counterparts in the final parse forest.

We have applied different strategies so as to compare their precisions. The results on $G^{T>N}$ and G^{TIG} are summed up in Table 3. These results give several valuable results. First, as we expected, the basic b-filter drastically reduces the size of the grammar. The result is even better on $G^{T>N}$ thanks to its large number of terminal symbols. Second, both the adjacency a-filter and the DSA d-filter efficiently reduce the size of the grammar: on $G^{T>N}$, the a-filter eliminates 20% of the productions they receive as input, a bit less for the d-filter. Indeed, the a-filter performs better than the d-filter introduced in (Boullier, 2003), at least as precision is concerned. We shall see later that this is still the case on global parsing times. However, applying the d-filter after the a-filter still removes a non-neglectible amount of productions:[17] each technique is able to eliminate productions that are kept by the other one. The result of these filters is suprisingly good: in average, after all filters, only approx. 20% of the productions that have been kept will not be successfully instantiated in the final parse forest. Third, the adjacency filter can be used in its one-pass mode, since almost all the benefit from the full (fix-point) mode is already reached after the first application. This is practically a very valuable result, since the one-pass mode is obviously faster than the full mode.

Strategy	Average precision	
	$G^{T>N}$	G^{TIG}
no filter	0.04%	0.03%
b	62.87%	39.43%
bd	74.53%	66.56%
ba	77.31%	66.94%
ba^∞	77.48%	67.48%
bad	80.27%	77.16%
$ba^\infty d$	80.30%	77.41%
gold	*100%*	*100%*

Table 3: Average precision of six different filtering strategies on our test corpus with $G^{T>N}$ and G^{TIG}.

However, all these filters do require computing time, and it is necessary to evaluate not only the precision of these filters, but also their execution time as well as the influence they have on the global (including filtering) parsing time .

4.3 Parsing time and best filter

Filter execution times for the six filtering strategies introduced in Table 3 are illustrated for $G^{T>N}$ in Figure 1. These graphics show three extremely valuable pieces of information. First, filtering times are extremely low: the average filtering time for the slowest filter ($ba^\infty d$, i.e., basic plus full adjacency plus DSA) on 40-word sentences is around 20 ms. Second, on small sentences, filtering times are virtually zero. This is important, since it means that there

[15]As seen above, inflected forms are directly terminal symbols of $G^{T>N}$, while G^{TIG} uses a *lexicon* to map these inflected forms into its own terminal symbols, thereby possibly introducing lexical ambiguity.

[16]Approx. 15% of the original set of sentences were not recognized, and required error recovery techniques; we decided to discard them for this experiment.

[17]Although not reported here, applying the a before d leads to the same conclusion.

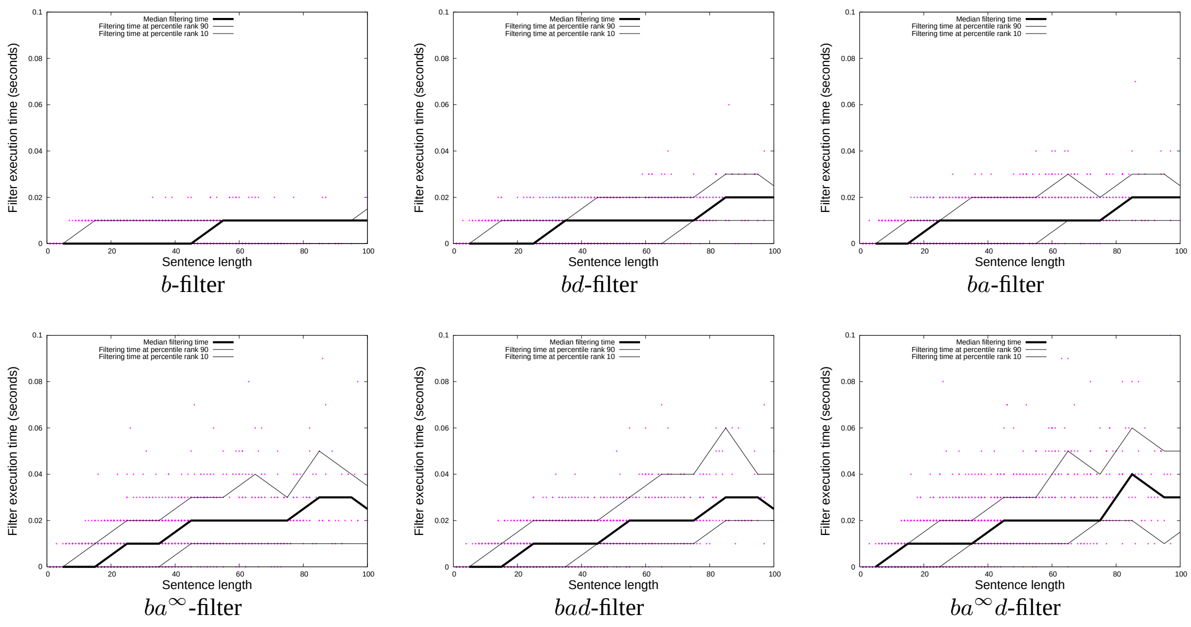


Figure 1: Filtering times for six different strategies with $G^{T>N}$

is almost no fixed cost to pay when we use these filters (let us recall that without any filter, building efficient parsers for such a huge grammar is highly problematic). Third, all these filters, at least when used with $G^{T>N}$, are executed in a time which is *linear* w.r.t. the size of the input sentence (i.e., the size of the input DAG).

The results on G^{TIG} lead to the same conclusions, with one exception: with this extremely huge grammar with so long right-hand sides, the basic filter is not as fast as on $G^{T>N}$ (and not as precise, as we will see below, which slows down the *make-a-reduced-grammar* algorithm since it is applied on a larger filtered grammars). For example, the median execution time for the basic filter on sentences whose size is approximately 40 is 0.25 seconds, to be compared with the 0.00 seconds reached on $G^{T>N}$ (this zero value means a median time strictly lower than 0.01 seconds, which is the granularity of our time measurments).

Figure 2 and 3 show the global (filtering+parsing) execution time for the 6 different filters. We only show median times computed on classes of sentences of length $10i$ to $10(i+1)-1$ and plotted with a centered x-coordinate $(10(i+1/2))$, but results with other percentiles or average times on the same classes draw the same overall picture.

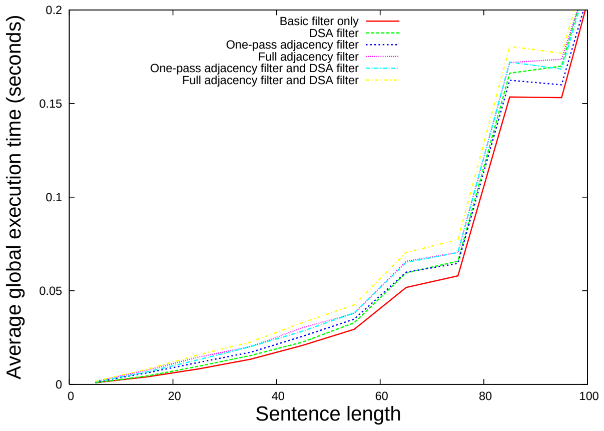


Figure 2: Global (filtering+parsing) times for six different strategies with $G^{T>N}$

One can see that the results are completely different, showing a strong dependency on the characteristics of the grammar. In the case of $G^{T>N}$, the huge number of terminal symbols and the reasonable average size of right-hand sides of productions, the basic filtering strategy is the best strategy: although it is fast because relatively simple, it reduces the grammar extremely efficiently (it has a 60.56% precision, to be compared with the precision of the void filter which is 0.04%). Hence, for $G^{T>N}$, our only result

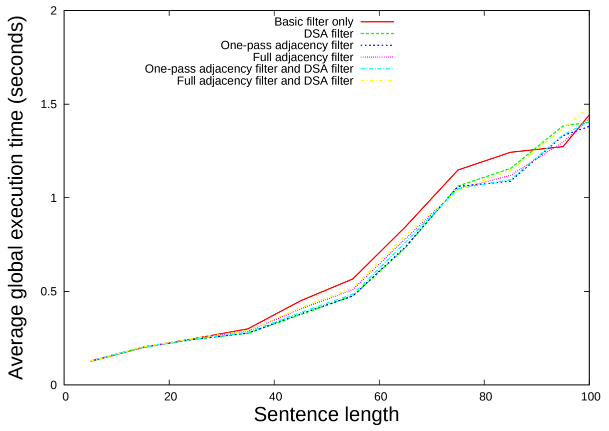


Figure 3: Global (filtering+parsing) times for six different strategies with G^{TIG}

is that this basic filter does allow us to build an efficient parser (the most efficient one), but that refined additionnal filtering strategies are not useful.

The picture is completely different with G^{TIG}. Contrary to $G^{T>N}$, this grammar has comparatively very few terminal and non-terminal symbols, and very long right-hand sides. These two facts lead to a lower precision of the basic filter (39.43%), which keeps many more productions when applied on G^{TIG} than when applied on $G^{T>N}$, and leads, when applied alone, to the less efficient parser. This gives to the adjacency filter much more opportunity to improve the global execution time. However, the complexity of the grammar makes the construction of the DSA filter relatively costly despite its precision, leading to the following conclusion: on G^{TIG} (and probably on any grammar with similar characteristics), the best filtering strategy is the one-pass adjacency strategy. In particular, this leads to an improvement over the work of (Boullier, 2003) which only introduced the DSA filter. Incidentally, the extreme size of G^{TIG} leads to much higher parsing times, approximately 10 times higher than with $G^{T>N}$, which is consistent with the ratio between the sizes of both involved grammars.

5 Conclusion

It is a well known result in optimization techniques that the key to practically improve these processes is to reduce their search space. This is also the case in parsing and in particular in CF parsing.

Many parsers process their inputs from left to right but we can find in the literature other parsing strategies. In particular, in NLP, (van Noord, 1997) and (Satta and Stock, 1994) propose bidirectional algorithms. These parsers have the reputation to have a better efficiency than their left-to-right counterpart. This reputation is not only based upon experimental results (van Noord, 1997) but also upon mathematical arguments in (Nederhof and Satta, 2000). This is specially true when the productions of the CFG strongly depend on lexical information. In that case the parsing search space is reduced because the constraints associated to lexical elements are evaluated as early as possible. We can note that our filtering strategies try to reach the same purpose by a totally different mean: we reduce the parsing search space by eliminating as many productions as possible, including possibly non-lexicalized productions whose irrelevance to parse the current input can not be directly deduced from that input.

We can also remark that our results are not in contradiction with the claims of (Nederhof and Satta, 2000) in which they argue that “Earley algorithm and related standard parsing techniques [. . .] cannot be directly extended to allow left-to-right and correct-prefix-property parsing in acceptable time bound”. First, as already noted in Section 1, our method does not work for any large CFG. In order to work well, the first step of our basic strategy must filter out a great amount of (lexicalized) productions. To do that, it is clear that the set of terminals in the input text must select a small ratio of lexicalized productions. To give a more concrete idea we advocate that the selected productions produce roughly a grammar of *normal* size out of the large grammar. Second, our method as a whole clearly does not process the input text from left-to-right and thus does not enter in the categories studied in (Nederhof and Satta, 2000). Moreover, the authors bring strong evidences that in case of polynomial-time off-line compilation of the grammar, left-to-right parsing cannot be performed in polynomial time, independently of the size of the lexicon. Once again, if our filter pass is viewed as an off-line processing of the large input grammar, our output is not a compilation of the large grammar, but a (compilation of a) smaller grammar, specialized in (some abstractions of) the source text only. In other words their negative results do not necessarily apply to our specific case.

The experiment campaign as been conducted in using an Earley-like parser.[18] We have also successfuly tried the coupling of our filtering strategies with a CYK parser (Kasami, 1967; Younger, 1967) as post-processor. However the coupling with a GLR parser (See (Satta, 1992) for example) is perhaps more problematic since the time taken to build up the underlying nondeterministic LR automaton from the sub-grammar can be prohibitive.

Though no definitive answer can be made to the question asked in the title, we have shown that, in some cases, the answer is certainly *yes*.

References

Sylvie Billot and Bernard Lang. 1989. The structure of shared forests in ambiguous parsing. In *Meeting of the Association for Computational Linguistics*, pages 143–151.

Pierre Boullier and Benoît Sagot. 2005. Efficient and robust LFG parsing: SxLfg. In *Proceedings of IWPT'05*, pages 1–10, Vancouver, Canada.

Pierre Boullier. 2000. On TAG parsing. *Traitement Automatique des Langues (T.A.L.)*, 41(3):759–793.

Pierre Boullier. 2003. Guided Earley parsing. In *Proceedings of IWPT 03*, pages 43–54, Nancy, France.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Communication of the ACM*, 13(2):94–102.

Jeffrey D. Hopcroft and John E. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Mass.

Aravind Joshi. 1997. Parsing techniques. In *Survey of the state of the art in human language technology*, pages 351–356. Cambridge University Press, New York, NY, USA.

Tadao Kasami. 1967. An efficient recognition and syntax algorithm for context-free languages. Scientific Report AFCRL-65–758, Air Force Cambridge Research Laboratory, Bedford, Massachusetts, USA.

Robert C. Moore. 2000. Improved left-corner chart parsing for large context-free grammars. In *Proceedings of IWPT 2000*, pages 171–182, Trento, Italy. Revised version at `http://www.cogs.susx.ac.uk/lab/nlp/carroll/cfg-resources/iwpt2000-rev2.ps`.

Mark-Jan Nederhof and Giorgio Satta. 2000. Left-to-right parsing and bilexical context-free grammars. In *Proceedings of the first conference on North American chapter of the ACL*, pages 272–279, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Mark-Jan Nederhof. 1993. Generalized left-corner parsing. In *Proceedings of the sixth conference on European chapter of the ACL*, pages 305–314, Morristown, NJ, USA. ACL.

Benoît Sagot and Pierre Boullier. 2005. From raw corpus to word lattices: robust pre-parsing processing. In *Proceedings of L&TC 2005*, pages 348–351, Poznań, Poland.

Benoît Sagot, Lionel Clément, Éric Villemonte de La Clergerie, and Pierre Boullier. 2006. The Lefff 2 syntactic lexicon for french: architecture, acquisition, use. In *Proc. of LREC'06*.

Giorgio Satta and Oliviero Stock. 1994. Bidirectional context-free grammar parsing for natural language processing. *Artif. Intell.*, 69(1-2):123–164.

Giorgio Satta. 1992. Review of "generalized lr parsing" by masaru tomita. kluwer academic publishers 1991. *Comput. Linguist.*, 18(3):377–381.

Yves Schabes and Richard C. Waters. 1995. Tree insertion grammar: Cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Comput. Linguist.*, 21(4):479–513.

Yves Schabes, Anne Abeillé, and Aravind K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. In *Proceedings of the 12th International Conference on Comput. Linguist. (COLING'88)*, Budapest, Hungary.

Gertjan van Noord. 1997. An efficient implementation of the head-corner parser. *Comput. Linguist.*, 23(3):425–456.

Éric Villemonte de La Clergerie. 2005. From metagrammars to factorized TAG/TIG parsers. In *Proceedings of IWPT'05*, pages 190–191, Vancouver, Canada.

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.

[18]Contrarily to classical Earley parsers, its *predictor* phase uses a pre-computed structure which is roughly an LC relation. Note that this feature forces our filters to compute an LC relation on the generated sub-grammar. This also shows that LC parsers may also benefit from our filtering techniques.

Pomset mcfgs

Michael J Pan
University of California Los Angeles
mjpan@cs.ucla.edu

Abstract

This paper identifies two orthogonal dimensions of context sensitivity, the first being context sensitivity in concurrency and the second being structural context sensitivity. We present an example from natural language which seems to require both types of context sensitivity, and introduce partially ordered multisets (pomsets) mcfgs as a formalism which succintly expresses both.

Introduction

Researchers in computer science and formal language theory have separately investigated context sensitivity of languages, addressing disjoint dimensions of context sensitivity. Researchers in parallel computing have explored the addition of concurrency and free word order to context free languages, i.e. a concurrency context sensitivity (Gischer, 1981; Warmuth and Haussler, 1984; Pratt, 1985; Pratt, 1986; Lodaya and Weil, 2000). Computational linguistis have explored adding crossing dependency and discontinuous constituency, i.e. a structural context sensitivity (Seki et al., 1991; Vijay-Shanker et al., 1987; Stabler, 1996).

Research considering the combination of two dimensions of expressing context sensitivity have been sparse, e.g. (Becker et al., 1991), with research dedicated to this topic virtually nonexistent. Natural languages are not well expressed by either form of context sensitivity alone. For example, in Table 1, sentences 1-8 are valid, but 9, 10 are invalid constructions of Norwegian. In addition to the crossing dependency between the determiner and adverb phrase, this example can be described by either

	Derfor ga **Jens Kari kyllingen** tydeligvis ikke lenger kald
	Therefore gave **Jens Kari the chicken** evidently not longer cold
	Derfor ga **Jens Kari** tydeligvis **kyllingen** ikke lenger kald
	Derfor ga **Jens** tydeligvis **Kari kyllingen** ikke lenger kald
	Derfor ga **Jens** tydeligvis **Kari** ikke **kyllingen** lenger kald
	Derfor ga **Jens** tydeligvis **Kari** ikke lenger **kyllingen** kald
	Derfor ga **Jens** tydeligvis ikke lenger **Kari kyllingen** kald
	Derfor ga tydeligvis **Jens** ikke lenger **Kari kyllingen** kald
	Derfor ga tydeligvis ikke **Jens** lenger **Kari kyllingen** kald
*	Derfor ga **Jens** ikke tydeligvis **Kari** lenger **kyllingen** kald
*	Derfor ga **Jens** ikke tydeligvis **kyllingen** lenger **Kari** kald

Table 1: Bobaljik's paradox/shape conservation example

Bobaljik's paradox (Bobaljik, 1999), which asserts that relative ordering of clausal constituents are not unambiguously determined by the phrase structure, or shape conservation (Müller, 2000), i.e. that linear precedence is preserved despite movement operations. In other words, the two structurally context sensitive components (due to the crossing dependency between them) can be shuffled arbitrarily, leading to concurrent context sensitivity.

This paper proposes pomset mcfgs as a formalism for perspicuously expressing both types of context sensitivity. [1] The rest of the paper is organized as follows. Section 1 introduces pomsets, pomset operations, and pomset properties. Section 2 provides a definition of pomset mcfgs by extending the standard definition of mcfgs, defined over tuples of strings, to tuples of pomsets. Section 3 discusses pomset mcfg parsing.

[1]Other pomset based formalisms (Lecomte and Retore, 1995; Basten, 1997; Nederhof et al., 2003) have been limited to the use of pomsets in context free grammars only.

1 Pomsets

In this section, we define pomsets as a model for describing concurrency. A labelled partial order (LPO) is a 4 tuple (V, Σ, $\preceq$, μ) where V is a set of vertices, Σ is the alphabet, $\preceq$ is the partial order on the vertices, and μ is the labelling function μ:V$\rightarrow$ Σ. A pomset is a LPO up to isomorphism. The concatenation of pomsets p and q is defined as ;(p,q) = (V$_p\cup$V$_q$,$\Sigma_p \cup \Sigma_q$,$\preceq_p \cup \preceq_q$ $\cup$V$_p\times$V$_q$,$\mu_p \cup \mu_q$). The concurrency of pomsets p and q is defined as $\|$(p,q) = (V$_p\cup$V$_q$,$\Sigma_p \cup \Sigma_q$,$\preceq_p \cup \preceq_q$,$\mu_p \cup \mu_q$). Pomset isolation (ι) is observed only in the context of concurrency. The concurrence of an isolated pomset with another pomset is defined as $\|$(ιp,q) = ({v$_p$}$\cup$V$_q$,p$_\lambda$ $\cup$ Σ_q,$\preceq_q$,{(p$_\lambda$,v$_p$)}$\cup\mu_q$), where λp is the set of linearizations for p, and p$_\lambda$ is a function which returns an element of λp. Let $\|_i$ be a pomset concurrency operator restricted to an arity of i. Because concurrency is both associative and commutative, without isolation, $\|_m\|_n$ = $\|_n\|_m$ = $\|_{m+n}$, defeating any arity restrictions. Isolation allows us to restrict the arity of the concurrency operator, guaranteeing that in all linearizations of the pomset, the linearizations of the isolated subpomsets are contiguous.[2] A mildly concurrent operator ι $\|_n$, i.e. an n-concurrent operator, is a composite operator whose concurrency is isolated and restricted to an arity of n, such that it operates on at most n items concurrently.

2 Pomset mcfgs

There are many (structural) mildly context sensitive grammar formalisms, e.g. mcfg, lcfrs, mg, and they have been shown to be equivalent (Vijay-Shanker et al., 1987). In this section we construct mcfgs over pomsets (instead of strings) to define grammars with both types of context sensitivity.

A pomset mcfg G is a 7-tuple (Σ,N,O,P,F,R,S) such that Σ is a finite non-empty set of atoms, i.e. terminal symbols, N is a finite non-empty set of nonterminal symbols, where N$\cap\Sigma$=$\emptyset$, O is a set of valid pomset operators, P is a set of i-tuples of pomsets labelled by $\Sigma\cup$N, F is a finite set of pomset rewriting functions from tuples of elements of P into elements in P, F$\subseteq$\{ g:P^n $\rightarrow$P | n>0 \}, R is a finite set of rewrite rules which pair n-ary elements of F with n+1 nonterminals, and S$\in$N is the start symbol, and d(S) = 1.

This definition extends the standard mcfg definition (Seki et al., 1991), with two main differences. First, strings have been generalized to pomsets, i.e. P is a set of i-tuples of pomsets instead of i-tuples of strings. It follows that F, the set of functions, operate on tuples of pomsets instead of tuples of strings, and so forth. Second, pomset mcfgs explicitly specify O, the set of possible operators over the pomsets, e.g. {;, ι $\|_2$}; string mcfgs have an implied operator set O={;} (i.e. just string concatenation).

Additionally, just as in mcfgs, where the arity of string components are limited, we can limit the arity of the concurrency of pomsets. A n-concurrent pomset mcfg is a pomset mcfg such that for all concurrency operators $\|_i$ in the grammar, i$\leq$n. A pomset mcfg with no concurrency among its components is a 1-concurrent pomset mcfg, just as a cfg is a 1-mcfg.

3 Parsing

In this section we propose a strategy for parsing pomset mcfgs, based on IDL parsing (Nederhof and Satta, 2004). We define pomset graphs, which extend IDL graphs and pom-automata and are defined over tuples of pomsets (or tuples of idl expressions), rather than single pomsets or idl expressions. An informal analysis of the computational complexity for parsing pomset mcfgs follows.

Pomset graphs The construction is quite straight forward, as pomsets themselves can already be considered as DAGs. However, in the pomset graph, we add two vertices, the start and end vertices. We then add precedence relations such that the start vertex precedes all minimal vertices of the pomset, and that the end vertex succeeds all maximal vertices of the pomset. For any nonempty pomset, we define V$_{min}$ $\subseteq$V and V$_{max}$ $\subseteq$V to be the minimal and maximal, respectively, vertices of V. Informally, no vertex in a pomset precede V$_{min}$ and none succeed any in V$_{max}$. Formally, $\forall$ v$\in$V, v'$\in$V,v'$\neq$v, V$_{min}$ = \{ v | (v',v) $\notin\preceq$ \} and V$_{max}$ = \{ v | (v,v') $\notin\preceq$ \}. The start vertex is then labelled with the empty string, ϵ, and the end vertex is labelled with σ', a symbol not in Σ.

[2]Pomset isolation is similar to proposals in for string isolation in linear specification language (Goetz and Penn, 2000), locking in idl-expressions (Nederhof and Satta, 2004), and integrity constraints in fo-tag (Becker et al., 1991).

Given a pomset p= (V_p,Σ,$\preceq$,μ_p), a pomset graph for p is a vertex labelled graph γ(p) = (V_γ,E,μ_γ) where V_γ and E are a finite set of vertices and edges, where V_γ=$V_p\cup\{v_s,v_e\}$ and E= $\preceq \cup v_s\times V_{min}\cup V_{max}\times v_e$, Σ_γ=$\Sigma\cup\{\epsilon,\sigma'\}$, where σ' is a symbol not in Σ, and μ_γ=$\mu_p\cup\{(v_s,\epsilon),(v_e,\sigma')\}$ is the vertex labelling function. Having defined the pomset graph, we can apply the IDL parsing algorithm to the graph.

Complexity While the complexity of the membership problem for pomset languages in general is NP-complete (Feigenbaum et al., 1993), by restricting the context sensitivity of the pomset grammars, polynomial time complexity is achievable. The complexity of the parsing of IDL graphs is O(n^{3k}) (Nederhof and Satta, 2004) where k is the width of the graph, and the width is a measurement of the number of paths being traversed in parallel, i.e. the arity of the concurrent context sensitivity. Our intuition is that the parameterization of the complexity according to the number of parallel paths applies even when structural context sensitivity is added. Thus for a k-concurrent m-structural mcfg, we conjecture that the complexity is O(n^{3km}).

4 Conclusion

In this paper we identified two types of context sensitivity, and provided a natural language example which exhibits both types of context sensitivity. We introduced pomset mcfgs as a formalism for describing grammars with both types of context sensitivity, and outlined an informal proof of the its polynomial-time parsing complexity.

References

Twan Basten. 1997. Parsing partially ordered multisets. *International Journal of Foundations of Computer Science*, 8(4):379–407.

Tilman Becker, Aravind K. Joshi, and Owen Rambow. 1991. Long distance scrambling and tree adjoining grammars. In *Proceedings of EACL-91, the 5th Conference of the European Chapter of the Association for Computational Linguistics*.

Jonathan David Bobaljik. 1999. Adverbs: The hierarchy paradox. *Glot International*, 4.

Joan Feigenbaum, Jeremy A. Kahn, and Carsten Lund. 1993. Complexity results for pomset languages. *SIAM Journal of Discrete Mathematics*, 6(3):432–442.

Jay Gischer. 1981. Shuffle languages, Petri nets, and context-sensitive grammars. *Communications of the ACM*, 24(9):597–605, September.

Thilo Goetz and Gerald Penn. 2000. A proposed linear specification language. Technical Report 134, Arbeitspapiere des SFB 340.

A. Lecomte and C. Retore. 1995. Pomset logic as an alternative categorial grammar. In Glyn Morrill and Richard Oehrle, editors, *Formal Grammar*, pages 181–196.

K. Lodaya and P. Weil. 2000. Series-parallel languages and the bounded-width property. *Theoretical Computer Science*, 237(1–2):347–380.

Gereon Müller. 2000. Shape conservation and remnant movement. In *Proceedings of NELS 30*.

Mark-Jan Nederhof and Giorgio Satta. 2004. IDL-expressions: A formalism for representing and parsing finite languages in natural language processing. *Journal of Artificial Intelligence Research*, 21:287–317.

Mark-Jan Nederhof, Giorgio Satta, and Stuart M. Shieber. 2003. Partially ordered multiset context-free grammars and ID/LP parsing. In *Proceedings of the Eighth International Workshop on Parsing Technologies*, pages 171–182, Nancy, France, April.

Vaughan R. Pratt. 1985. The pomset model of parallel processes : Unifying the temporal and the spatial. Technical report, Stanford University, January.

Vaughan R. Pratt. 1986. Modelling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71.

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context free grammars. *Theoretical Computer Science*, 88:191–229.

Edward P. Stabler. 1996. Derivational minimalism. In Christian Retoré, editor, *LACL*, volume 1328 of *Lecture Notes in Computer Science*, pages 68–95. Springer.

K. Vijay-Shanker, D. J. Weir, and A. K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the ACL*, pages 104–111, Stanford, CA.

Manfred K. Warmuth and David Haussler. 1984. On the complexity of iterated shuffle. *J. Comput. Syst. Sci.*, 28(3):345–358.

Modular and Efficient Top-Down Parsing for Ambiguous Left-Recursive Grammars

Richard A. Frost and Rahmatullah Hafiz
School of Computer Science
University of Windsor
Canada
rfrost@cogeco.ca

Paul C. Callaghan
Department of Computer Science
University of Durham
U.K.
P.C.Callaghan@durham.ac.uk

Abstract

In functional and logic programming, parsers can be built as modular executable specifications of grammars, using parser combinators and definite clause grammars respectively. These techniques are based on top-down backtracking search. Commonly used implementations are inefficient for ambiguous languages, cannot accommodate left-recursive grammars, and require exponential space to represent parse trees for highly ambiguous input. Memoization is known to improve efficiency, and work by other researchers has had some success in accommodating left recursion. This paper combines aspects of previous approaches and presents a method by which parsers can be built as modular and efficient executable specifications of ambiguous grammars containing unconstrained left recursion.

1 Introduction

Top-down parsers can be built as a set of mutually-recursive processes. Such implementations are modular in the sense that parsers for terminals and simple non-terminals can be built and tested first. Subsequently, parsers for more complex non-terminals can be constructed and tested. Koskimies (1990), and Nederhof and Koster (1993) discuss this and other advantages of top-down parsing.

In functional and logic programming, top-down parsers can be built using parser combinators (e.g. see Hutton 1992 for a discussion of the origins of parser combinators, and Frost 2006 for a discussion of their use in natural-language processing) and definite clause grammars (DCGs) respectively. For example, consider the following grammar, in which `s` stands for sentence, `np` for nounphrase, `vp` for verbphrase, and `det` for determiner:

```
s    ::= np vp
np   ::= noun | det noun
vp   ::= verb np
det  ::= 'a' | 't'
noun ::= 'i' | 'm' | 'p' | 'b'
verb ::= 's'
```

A set of parsers for this grammar can be constructed in the Haskell functional programming language as follows, where `term`, `‘orelse‘`, and `‘thenS‘` are appropriately-defined higher-order functions called parser combinators. (Note that backquotes surround infix functions in Haskell).

```
s    = np ‘thenS‘ vp
np   = noun ‘orelse‘ (det ‘thenS‘ noun)
vp   = verb ‘thenS‘ np
det  = term 'a' ‘orelse‘ term 't'
noun = term 'i' ‘orelse‘ term 'm'
                ‘orelse‘ term 'p'
                ‘orelse‘ term 'b'
verb = term 's'
```

Note that the parsers are written directly in the programming language, in code which is similar in structure to the rules of the grammar. As such, the implementation can be thought of as an executable specification with all of the associated advantages. In particular, this approach facilitates modular piecewise construction and testing of component parsers. It also allows parsers to be defined to return semantic values directly instead of intermediate parse results, and parsers to be parameterized in order to accommodate context-sensitive languages (e.g. Eijck 2003). Also, in functional programming, the type checker can be used to catch errors in parsers attributed with semantic actions.

Parser combinators and DCGs have been used extensively in applications such as prototyping of compilers, and the creation of natural language interfaces to databases, search engines, and web pages, where complex and varied semantic actions are closely integrated with syntactic processing. However, both techniques are based on top-down recursive descent search with backtracking. Commonly used implementations have exponential complexity for ambiguous languages, cannot handle left-recursion, and do not produce compact representations of parse trees. (Note, a left-recursive grammar is one in which a non-terminal `p` derives an expansion `p ..` headed with a `p` either directly or indirectly. Application of a parser for such a grammar results in infinite descent.) These shortcomings limit the use of parser combinators and DCGs especially in natural-language processing.

The problem of exponential time complexity in top-down parsers constructed as sets of mutually-recursive functions has been solved by Norvig (1991) who uses memotables to achieve polynomial complexity. Norvig's technique is similar to the use of dynamic programming and state sets in Earley's algorithm (1970), and tables in the CYK algorithm of Cocke, Younger and Kasami. The basic idea in Norvig's approach is that when a parser is applied to the input, the result is stored in a memotable for subsequent reuse if the same parser is ever reapplied to the same input. In the context of parser combinators, Norvig's approach can be implemented using a function `memoize` to selectively "memoize" parsers.

In some applications, the problem of left-recursion can be overcome by transforming the grammar to a weakly equivalent non-left-recursive form. (i.e. to a grammar which derives the same set of sentences). Early methods of doing this resulted in grammars that are significantly larger than the original grammars. This problem of grammar size has been solved by Moore (2000) who developed a method, based on a left-corner grammar transformation, which produces non-left recursive grammars that are not much larger than the originals. However, although converting a grammar to a weakly-equivalent form is appropriate in some applications (such as speech recognition) it is not appropriate in other applications. According to Aho, Sethi, and Ullman (1986) converting a grammar to non-left recursive form makes it harder to translate expressions containing left-associative operators. Also, in NLP it is easier to integrate semantic actions with parsing when both leftmost and rightmost parses of ambiguous input are being generated. For example, consider the first of the following grammar rules:

```
np   ::= noun  | np conj np
conj ::= "and" | "or"
noun ::= "jim" | "su" | "ali"
```

and its non-left-recursive weakly equivalent form:

```
np  ::= noun np'
np' ::= conj np np' | empty
```

The non-left-recursive form loses the leftmost parses generated by the left-recursive form. Integrating semantic actions with the non-left-recursive rule in order to achieve the two correct interpretations of input such as `["john", "and", "su", "or", "ali"]` is significantly harder than with the left-recursive form.

Several researchers have recognized the importance of accommodating left-recursive grammars in top-down parsing, in general and in the context of parser combinators and DCGs in particular, and have proposed various solutions. That work is described in detail in section 3.

In this paper, we integrate Norvig's technique with aspects of existing techniques for dealing with left recursion. In particular: a) we make use of the length of the remaining input as does Kuno (1965), b) we keep a record of how many times each parser is applied to each input position in a way that is similar to the use of cancellation sets by Nederhof and Koster (1993), c) we integrate memoization with a technique for dealing with left recursion as does Johnson (1995), and d) we store "left-recursion counts" in the memotable, and encapsulate the memoization process in a programming construct called a monad, as suggested by Frost and Hafiz (2006).

Our method includes a new technique for accommodating indirect left recursion which ensures correct reuse of stored results created through curtailment of left-recursive parsers. We also modify the memoization process so that the memotable represents the potentially exponential number of parse trees in a compact polynomial sized form using a technique derived from the chart parsing methods of Kay (1980) and Tomita (1986).

As an example use of our method, consider the following ambiguous left-recursive grammar from Tomita (1985) in which pp stands for prepositional phrase, and prep for preposition. This grammar is left recursive in the rules for s and np. Experimental results using larger grammars are given later.

```
s    ::= np vp | s pp
np   ::= noun | det noun | np pp
pp   ::= prep np
vp   ::= verb np
det  ::= ’a’ | ’t’
noun ::= ’i’ | ’m’ | ’p’ | ’b’
verb ::= ’s’
prep ::= ’n’ | ’w’
```

The Haskell code below defines a parser for the above grammar, using our combinators:

```
s    = memoize "s"  ((np ‘thenS‘ vp)
              ‘orelse‘ (s ‘thenS‘ pp))
np   = memoize "np" (noun
            ‘orelse‘ (det ‘thenS‘ noun)
            ‘orelse‘ (np  ‘thenS‘ pp))
pp   = memoize "pp"   (prep ‘thenS‘ np)
vp   = memoize "vp"   (verb ‘thenS‘ np)
det  = memoize "det"  (term ’a’
                    ‘orelse‘ term ’t’)
noun = memoize "noun" (term ’i’
                    ‘orelse‘ term ’m’
                    ‘orelse‘ term ’p’
                    ‘orelse‘ term ’b’)
verb = memoize "verb" (term ’s’)
prep = memoize "prep" (term ’n’
                    ‘orelse‘ term ’w’)
```

The following shows the output when the parser function s is applied to the input string "isamntpwab", representing the sentence “I saw a man in the park with a bat”. It is a compact representation of the parse trees corresponding to the several ways in which the whole input can be parsed as a sentence, and the many ways in which subsequences of it can be parsed as nounphrases etc. We discuss this representation in more detail in subsection 4.4.

```
apply s "isamntpwab" =>

"noun"
  1  ((1,2),   [Leaf "i"])
  4  ((4,5),   [Leaf "m"])
  7  ((7,8),   [Leaf "p"])
 10  ((10,11), [Leaf "b"])
"det"
  3  ((3,4),   [Leaf "a"])
  6  ((6,7),   [Leaf "t"])
  9  ((9,10),  [Leaf "a"])
"np"
  1 ((1,2), [SubNode ("noun", (1,2))])
  3 ((3,5), [Branch [SubNode ("det", (3,4)),
                     SubNode ("noun",(4,5))]])
    ((3,8), [Branch [SubNode ("np",  (3,5)),
                     SubNode ("pp",  (5,8))]])
    ((3,11),[Branch [SubNode ("np",  (3,5)),
                     SubNode ("pp",  (5,11))],
             Branch [SubNode ("np",  (3,8)),
                     SubNode ("pp",  (8,11))]])
  6 ((6,8), [Branch [SubNode ("det", (6,7)),
                     SubNode ("noun",(7,8))]])
    ((6,11),[Branch [SubNode ("np",  (6,8)),
                     SubNode ("pp",  (8,11))]])
  9 ((9,11),[Branch [SubNode ("det", (9,10)),
                     SubNode ("noun",(10,11))]])
"prep"
  5 ((5,6), [Leaf "n"])
  8 ((8,9), [Leaf "w"])
"pp"
  8 ((8,11),[Branch [SubNode ("prep",(8,9)),
                     SubNode ("np",  (9,11))]])
  5 ((5,8), [Branch [SubNode ("prep",(5,6)),
                     SubNode ("np",  (6,8))]])
    ((5,11),[Branch [SubNode ("prep",(5,6)),
                     SubNode ("np",  (6,11))]])
"verb"
  2 ((2,3), [Leaf "s"])
"vp"
  2 ((2,5), [Branch [SubNode ("verb",(2,3)),
                     SubNode ("np",  (3,5))]])
    ((2,8), [Branch [SubNode ("verb",(2,3)),
                     SubNode ("np",  (3,8))]])
    ((2,11),[Branch [SubNode ("verb",(2,3)),
                     SubNode ("np",  (3,11))]])
"s"
  1 ((1,5), [Branch [SubNode ("np",  (1,2)),
                     SubNode ("vp",  (2,5))]])
    ((1,8), [Branch [SubNode ("np",  (1,2)),
                     SubNode ("vp",  (2,8))],
             Branch [SubNode ("s",   (1,5)),
                     SubNode ("pp",  (5,8))]])
    ((1,11),[Branch [SubNode ("np",  (1,2)),
                     SubNode ("vp",  (2,11))],
             Branch [SubNode ("s",   (1,5)),
                     SubNode ("pp",  (5,11))],
             Branch [SubNode ("s",   (1,8)),
                     SubNode ("pp",  (8,11))]]
```

Our method has two disadvantages: a) it has O(n^4) time complexity, for ambiguous grammars, compared with O(n^3) for Earley-style parsers (Earley 1970), and b) it requires the length of the input to be known before parsing can commence.

Our method maintains all of the advantages of top-down parsing and parser combinators discussed earlier. In addition, our method accommodates arbitrary context-free grammars, terminates correctly and correctly reuses results generated by direct and indirect left recursive rules. It parses ambiguous languages in polynomial time and creates polynomial-sized representations of parse trees.

In many applications the advantages of our approach will outweigh the disadvantages. In particular, the additional time required for parsing will not be a major factor in the overall time required when semantic processing, especially of ambiguous input, is taken into account.

We begin with some background material, showing how our approach relates to previous work by others. We follow that with a detailed description of our method. Sections 5, 6, and 7 contain informal proofs of termination and complexity, and a brief description of a Haskell implementation of our algorithm. Complete proofs and the Haskell code are available from any of the authors.

We tested our implementation on four natural-language grammars from Tomita (1986), and on four abstract highly-ambiguous grammars. The results, which are presented in section 8, indicate that our method is viable for many applications, especially those for which parser combinators and definite clause grammars are particularly well-suited.

We present our approach with respect to parser combinators. However, our method can also be implemented in other languages which support recursion and dynamic data structures.

2 Top-Down Backtracking Recognition

Top-down recognizers can be implemented as a set of mutually recursive processes which search for parses using a top-down expansion of the grammar rules defining non-terminals while looking for matches of terminals with tokens on the input. Tokens are consumed from left to right. Backtracking is used to expand all alternative right-hand-sides of grammar rules in order to identify all possible parses. In the following we assume that the input is a sequence of tokens `input`, of length `l_input` the members of which are accessed through an index `j`. Unlike commonly-used implementations of parser combinators, which produce recognizers that manipulate subsequences of the input, we assume, as in Frost and Hafiz (2006), that recognizers are functions which take an index `j` as argument and which return a set of indices as result. Each index in the result set corresponds to the position at which the recognizer successfully finished recognizing a sequence of tokens that began at position `j` . An empty result set indicates that the recognizer failed to recognize any sequence beginning at j. Multiple results are returned for ambiguous input.

According to this approach, a recognizer `term_t` for a terminal `t` is a function which takes an index `j` as input, and if `j` is greater than `l_input`, the recognizer returns an empty set. Otherwise, it checks to see if the token at position `j` in the input corresponds to the terminal `t`. If so, it returns a singleton set containing `j + 1`, otherwise it returns the empty set. For example, a basic recognizer for the terminal `'s'` can be defined as follows (note that we use a functional pseudo code throughout, in order to make the paper accessible to a wide audience. We also use a list lookup offset of 1):

```
term_s     = term 's'
where term t  j
 = {}     , if j > l_input
 = {j + 1}, if jth element of input = t
 = {}     , otherwise
```

The `empty` recognizer is a function which always succeeds returning its input index in a set:

```
empty j = {j}
```

A recognizer corresponding to a construct `p | q` in the grammar is built by combining recognizers for `p` and `q`, using the parser combinator `‘orelse‘`. When the composite recognizer is applied to index `j`, it applies `p` to `j`, then it applies `q` to `j`, and subsequently unites the resulting sets.:

```
(p ‘orelse‘ q) j = unite (p j) (q j)
```

e.g, assuming that the input is `"ssss"`, then

```
(empty ‘orelse‘ term_s) 2 => {2, 3}
```

A composite recognizer corresponding to a sequence of recognizers `p q` on the right hand side of a grammar rule, is built by combining those recognizers using the parser combinator `‘thenS‘`. When the composite recognizer is applied to an index `j`, it first applies `p` to `j`, then it applies `q` to each index in the set of results returned by `p`. It returns the union of these applications of `q`.

```
(p ‘thenS‘ q) j = union (map q (p j))
```

e.g., assuming that the input is `"ssss"`, then

```
(term_s ‘thenS‘ term_s) 1 => {3}
```

The combinators above can be used to define composite mutually-recursive recognizers. For example, the grammar `sS ::= 's' sS sS | empty` can be encoded as follows:

```
sS = (term_s ‘thenS‘ sS ‘thenS‘ sS)
     ‘orelse‘ empty
```

Assuming that the input is `"ssss"`, the recognizer `sS` returns a set of five results, the first four corresponding to proper prefixes of the input being recognized as an `sS`. The result `5` corresponds to the case where the whole input is recognized as an `sS`.

```
sS   1  => {1, 2, 3, 4, 5}
```

The method above does not terminate for left-recursive grammars, and has exponential time complexity with respect to `l_input` for non-left-recursive grammars. The complexity is due to the fact that recognizers may be repeatedly applied to the same index during backtracking induced by the operator `'orelse'`. We show later how complexity can be improved, using Norvig's memoization technique. We also show, in section 4.4, how the combinators `term`, `'orelse'`, and `'thenS'` can be redefined so that the processors create compact representations of parse trees in the memotable, with no effect on the form of the executable specification.

3 Left Recursion and Top-Down Parsing

Several researchers have proposed ways in which left-recursion and top-down parsing can coexist:

1) Kuno (1965) was the first to use the length of the input to force termination of left-recursive descent in top-down parsing. The minimal lengths of the strings generated by the grammar on the continuation stack are added and when their sum exceeds the length of the remaining input, expansion of the current non-terminal is terminated. Dynamic programming in parsing was not known at that time, and Kuno's method has exponential complexity.

2) Shiel (1976) recognized the relationship between top-down parsing and the use of state sets and tables in Earley and SYK parsers and developed an approach in which procedures corresponding to non-terminals are called with an extra parameter indicating how many terminals they should read from the input. When a procedure corresponding to a non-terminal `n` is applied, the value of this extra parameter is partitioned into smaller values which are passed to the component procedures on the right of the rule defining `n`. The processor backtracks when a procedure defining a non-terminal is applied with the same parameter to the same input position. The method terminates for left-recursion but has exponential complexity.

3) Leermakers (1993) introduced an approach which accommodates left-recursion through "recursive ascent" rather than top-down search. Although achieving polynomial complexity through memoization, the approach no longer has the modularity and clarity associated with pure top-down parsing. Leermakers did not extend his method to produce compact representations of trees.

4) Nederhof and Koster (1993) introduced "cancellation" parsing in which grammar rules are translated into DCG rules such that each DCG non-terminal is given a "cancellation set" as an extra argument. Each time a new non-terminal is derived in the expansion of a rule, this non-terminal is added to the cancellation set and the resulting set is passed on to the next symbol in the expansion. If a non-terminal is derived which is already in the set then the parser backtracks. This technique prevents non-termination, but loses some parses. To solve this, for each non-terminal `n`, which has a left-recursive alternative 1) a function is added to the parser which places a special token $\underline{n}$ at the front of the input to be recognized, 2) a DCG corresponding to the rule `n ::=` $\underline{n}$ is added to the parser, and 3) the new DCG is invoked after the left-recursive DCG has been called. The approach accommodates left-recursion and maintains modularity. An extension to it also accommodates hidden left recursion which can occur when the grammar contains rules with empty right-hand sides. The shortcoming of Nederhof and Koster's approach is that it is exponential in the worst case and that the resulting code is less clear as it contains additional production rules and code to insert the special tokens.

5) Lickman (1995) defined a set of parser combinators which accommodate left recursion. The method is based on an idea by Philip Wadler in an unpublished paper in which he claimed that fixed points could be used to accommodate left recursion. Lickman implemented Wadler's idea and provided a proof of termination. The method accommodates left recursion and maintains modularity and clarity of the code. However, it has exponential complexity, even for recognition.

6) Johnson (1995) appears to have been the first to integrate memoization with a method for dealing with left recursion in pure top-down parsing. The basic idea is to use the continuation-passing style of programming (CPS) so that the parser computes multiple results, for ambiguous cases, incrementally. There appears to have been no attempt to extend Johnson's approach to create compact representations of parse trees. One explanation for this could

be that the approach is somewhat convoluted and extending it appears to be very difficult. In fact, Johnson states, in his conclusion, that "an implementation attempt (to create a compact representation) would probably be very complicated."

7) Frost and Hafiz (2006) defined a set of parser combinators which can be used to create polynomial time recognizers for grammars with direct left recursion. Their method stores left-recursive counts in the memotable and curtails parses when a count exceeds the length of the remaining input. Their method does not accommodate indirect left recursion, nor does it create parse trees.

Our new method combines many of the ideas developed by others: as with the approach of Kuno (1965) we use the length of the remaining input to curtail recursive descent. Following Shiel (1976), we pass additional information to parsers which is used to curtail recursion. The information that we pass to parsers is similar to the cancellation sets used by Nederhof and Koster (1993) and includes the number of times a parser is applied to each input position. However, in our approach this information is stored in a memotable which is also used to achieve polynomial complexity. Although Johnson (1995) also integrates a technique for dealing with left recursion with memoization, our method differs from Johnson's O(n^3) approach in the technique that we use to accommodate left recursion. Also, our approach facilitates the construction of compact representations of parse trees whereas Johnson's appears not to. In the Haskell implementation of our algorithm, we use a functional programming structure called a monad to encapsulate the details of the parser combinators. Lickman's (1995) approach also uses a monad, but for a different purpose. Our algorithm stores "left-recursion counts" in the memotable as does the approach of Frost and Hafiz (2006). However, our method accommodates indirect left recursion and can be used to create parsers, whereas the method of Frost and Hafiz can only accommodate direct left recursion and creates recognizers not parsers.

4 The New Method

We begin by describing how we improve complexity of the recognizers defined in section 2. We then show how to accommodate direct and indirect left recursion. We end this section by showing how recognizers can be extended to parsers.

4.1 Memoization

As in Norvig (1991) a memotable is constructed during recognition. At first the table is empty. During the process it is updated with an entry for each recognizer r_i that is applied. The entry consists of a set of pairs, each consisting of an index `j` at which the recognizer r_i has been applied, and a set of results of the application of r_i to `j`.

The memotable is used as follows: whenever a recognizer r_i is about to be applied to an index `j`, the memotable is checked to see if that recognizer has ever been applied to that index before. If so, the results from the memotable are returned. If not, the recognizer is applied to the input at index `j`, the memotable is updated, and the results are returned. For non-left-recursive recognizers, this process ensures that no recognizer is ever applied to the same index more than once.

The process of memoization is achieved through the function `memoize` which is defined as follows, where the `update` function stores the result of recognizer application in the table:

```
memoize label r_i j
 = if lookup label j succeeds,
       return memotable result
   else apply r_i to j,
       update table, and return results
```

Memoized recognizers, such as the following, have cubic complexity (see later):

```
msS = memoize "msS"((ms ‘thenS‘ msS
                         ‘thenS‘ msS)
                    ‘orelse‘ empty)
 ms = memoize "ms" term_s
```

4.2 Accommodating direct left recursion

In order to accommodate direct left recursion, we introduce a set of values c_ij denoting the number of times each recognizer r_i has been applied to the index `j`. For non-left-recursive recognizers this "left-rec count" will be at most one, as the memotable lookup will prevent such recognizers from ever being applied to the same input twice. However, for left-recursive recognizers, the left-rec count is increased on recursive descent (owing to the fact that the memotable is only updated on recursive ascent after the recognizer has been applied). Application of a recognizer r to an index j is failed whenever the left-rec count exceeds the number of unconsumed tokens of the input plus 1. At this point no parse is possible (other than spurious parses which could occur with circular grammars — which we want to reject). As illustration, consider the following branch being created during the parse of two remaining tokens on the input (where N, P and Q are nodes in the parse search space corresponding to non-terminals, and A, B and C to terminals or non-terminals):

```
              N
             / \
            N   A
           / \
          N   B
         / \
        P   C
       /
      Q
     /
    N
```

The last call of the parser for N should be failed owing to the fact that, irrespective of what A, B, and C are, either they must require at least one input token, otherwise they must rewrite to empty. If they all require a token, then the parse cannot succeed. If any of them rewrite to empty, then the grammar is circular (N is being rewritten to N) and the last call should be failed in either case.

Note that failing a parse when a branch is longer than the length of the remaining input is incorrect as this can occur in a correct parse if recognizers are rewritten into other recognizers which do not have "token requirements to the right". For example, we cannot fail the parse at P or Q as these could rewrite to empty without indicating circularity. Also note that we curtail the recognizer when the left-rec count exceeds the number of unconsumed tokens *plus 1*. The plus 1 is necessary to accommodate the case where the recognizer rewrites to empty on application to the end of the input.

To make use of the left-rec counts, we simply modify the memoize function to refer to an additional table called ctable which contains the left-rec counts c_ij, and to check and increment these counters at appropriate points in the computation: if the memotable lookup for the recognizer r_i and the index j produces a result, that result is returned. However, if the memotable does not contain a result for that recognizer and that index, c_ij is checked to see if the recognizer should be failed because it has descended too far through left-recursion. If so, memoize returns an empty set as result with the memotable unchanged. Otherwise, the counter c_ij is incremented and the recognizer r_i is applied to j, and the memotable is updated with the result before it is returned. The function memoize defined below, can now be applied to rules with direct left recursion.

```
memoize label r_i j  =
 if lookup label j succeeds
    return memotable results
 else if c_ij > (l_input)-j+1, return {}
     else increment c_ij, apply r_i to j,
          update memotable,
              and return results
```

4.3 Accommodating indirect left recursion

We begin by illustrating how the method described above may return incomplete results for grammars containing indirect left recursion.

Consider the following grammar, and subset of the search space, where the left and right branches represent the expansions of the first two alternate right-hand-sides of the rule for the non terminal S, applied to the same position on the input:

```
S ::= S then ..| Q | P | x       S
P ::= S then .                 /    \
Q ::= T                   S then ..  Q
T ::= P                   |          |
                          S then ..  T
                          |          |
                          P          P
                          |          |
                        S then..  S then ..
                         |
                       fail S
```

Suppose that the left branch occurs before the right branch, and that the left branch was failed due to the left-rec count for S exceeding its limit. The results stored for P on recursive ascent of the left branch would be an empty set. The problem is that the later call of P on the right branch should not reuse the empty set of results from the first call of P as they are incomplete with respect to the position of P on the right branch (i.e. if P were to be re-applied to the input in the context of the right branch, the results would not necessarily be an empty set). This problem is a result of the fact that S caused curtailment of the results for P as well as for itself. This problem can be solved as follows:

1) Pass left-rec contexts down the parse space. We need additional information when storing and considering results for reuse. We begin by defining the "left-rec-context" of a node in the parse search space as a list of the following type, containing for each index, the left-rec count for each recognizer, including the current recognizer, which have been called in the search branch leading to that node:

`[(index,[(recog_label,left_rec_count)])]`

2) Generate the reasons for curtailment when computing results. For each result we need to know if the subtrees contributing to it have been curtailed through a left-rec limits, and if so, which recognizers, at which indices, caused the curtailment. A list of `(recog_label, index)` pairs which caused curtailment in any of the subtrees is returned with the result. `'orelse'` and `'thenS'` are modified, accordingly, to merge these lists, in addition to merging the results from subtrees.

3) Store results in the memotable together with a subset of the current left-rec context corresponding to those recognizers which caused the curtailment. When a result is to be stored in the memotable for a recognizer `P`, the list of recognizers which caused curtailment (if any) in the subtrees contributing to this result is examined. For each recognizer `S` which caused curtailment at some index, the current left-rec counter for `S` at that index (in the left-rec context for `P`) is stored with the result for `P`. This means that the only part of the left-rec context of a node, that is stored with the result for that node, is a list of those recognizers and current left-rec counts which had an effect on curtailing the result. The limited left-rec context which is stored with the result is called the "left-rec context of the result".

4) Consider results for reuse. Whenever a memotable result is being considered for reuse, the left-rec-context of that result is compared with the left-rec-context of the current node in the parse search. The result is only reused if, for each recognizer and index in the left-rec context of the result, the left-rec-count is smaller than or equal to the left-rec-count of that recognizer and index in the current context. This ensures that a result stored for some application `P` of a recognizer at index `j` is only reused by a subsequent application `P'` of the same recognizer at the same position, if the left-rec context for `P'` would constrain the result more, or equally as much, as it had been constrained by the left-rec context for `P` at `j`. If there were no curtailment, the left-rec context of a result would be empty and that result can be reused anywhere irrespective of the current left-rec context.

4.4 Extending recognizers to parsers

Instead of returning a list of indices representing successful end points for recognition, parsers also return the parse trees. However, in order that these trees be represented in a compact form, they are constructed with reference to other trees that are stored in the memotable, enabling the explicit sharing of common subtrees, as in Kay's (1980) and Tomita's (1986) methods. The example in section 1 illustrates the results returned by a parser.

Parsers for terminals return a leaf value together with an endpoint, stored in the memotable as illustrated below, indicating that the terminal `"s"` was identified at position 2 on the input:

```
"verb" 2 ((2,3),[Leaf "s"])
```

The combinator `'thenS'` is extended so that parsers constructed with it return parse trees which are represented using reference to their immediate subtrees. For example:

```
"np" ......
3 ((3,5),[Branch[SubNode("det", (3,4)),
                 SubNode("noun",(4,5))]])
```

This memotable entry shows that a parse tree for a nounphrase `"np"` has been identified, starting at position `3` and finishing at position `5`, and which consists of two subtrees, corresponding to a determiner and a noun.

The combinator `'orelse'` unites results from two parsers and also groups together trees which have the same begin and end points. For example:

```
"np" .........
3 ((3,5),[Branch[SubNode("det", (3,4)),
                 SubNode("noun",(4,5))]])
 ((3,8), [Branch[SubNode("np",  (3,5)),
                 SubNode("pp",  (5,8))]])
 ((3,11),[Branch[SubNode("np",  (3,5)),
                 SubNode("pp",  (5,11))],
          Branch[SubNode("np",  (3,8)),
                 SubNode("pp",  (8,11))]])
```

which shows that four parses of a nounphrase "np" have been found starting at position 3, two of which share the endpoint 11.

An important feature is that trees for the same syntactic category having the same start/end points are grouped together and it is the group that is referred to by other trees of which it is a constituent. For example, in the following the parse tree for a "vp" spanning positions 2 to 11 refers to a group of subtrees corresponding to the two parses of an "np" both of which span positions 3 to 11:

```
"vp" 2 (["np"],[])
 ((2,5), [Branch[SubNode("verb",(2,3)),
                 SubNode("np",  (3,5))]])
 ((2,8), [Branch[SubNode("verb",(2,3)),
                 SubNode("np",  (3,8))]])
 ((2,11),[Branch[SubNode("verb",(2,3)),
                 SubNode("np",  (3,11))]])
```

5 Termination

The only source of iteration is in recursive function calls. Therefore, proof of termination is based on the identification of a measure function which maps the arguments of recursive calls to a well-founded ascending sequence of integers.

Basic recognizers such as `term 'i'` and the recognizer `empty` have no recursion and clearly terminate for finite input. Other recognizers that are defined in terms of these basic recognizers, through mutual and nested recursion, are applied by the `memoize` function which takes a recognizer and an index `j` as input and which accesses the `memotable`. An appropriate measure function maps the index and the set of left–rec values to an integer, which increases by at least one for each recursive call. The fact that the integer is bounded by conditions imposed on the maximum value of the index, the maximum values of the left-rec counters, and the maximum number of left-rec contexts, establishes termination. Extending recognizers to parsers does not involve any additional recursive calls and consequently, the proof also applies to parsers. A formal proof is available from any of the authors.

6 Complexity

The following is an informal proof. A formal proof is available from any of the authors.

We begin by showing that memoized non-left-recursive and left-recursive recognizers have a worst-case time complexities of O(n^3) and O(n^4) respectively, where `n` is the number of tokens in the input. The proof proceeds as follows: `'orelse'` requires O(n) operations to merge the results from two alternate recognizers provided that the indices are kept in ascending order. `'then'` involves O(n^2) operations when applying the second recognizer in a sequence to the results returned by the first recognizer. (The fact that recognizers can have multiple alternatives involving multiple recognizers in sequence increases cost by a factor that depends on the grammar, but not on the length of the input). For non-left-recursive recognizers, `memoize` guarantees that each recognizer is applied at most once to each input position. It follows that non-left recursive recognizers have O(n^3) complexity. Recognizers with direct left recursion can be applied to the same input position at most `n` times. It follows that such recognizers have O(n^4) complexity. In the worst case a recognizer with indirect left recursion could be applied to the same input position `n * nt` times where `nt` is the number of nonterminals in the grammar. This worst case would occur when every nonterminal was involved in the path of indirect recursion for some nonterminal. Complexity remains O(n^4).

The only difference between parsers and recognizers is that parsers construct and store parts of parse trees rather than end points. We extend the complexity analysis of recognizers to that of parsers and show that for grammars in Chomsky Normal Form (CNF) (i.e. grammars whose right-hand-sides have at most two symbols, each of which can be either a terminal or a non-terminal), the complexity of non-left recursive parsers is O(n^3) and of left-recursive parsers it is O(n^4). The analysis begins by defining a "parse tuple" consisting of a parser name `p`, a start/end point pair `(s, e)`, and a list of parser names and end/point pairs corresponding to the first level of the parse tree returned by `p` for the sequence of tokens from `s` to `e`. (Note that this corresponds to an entry in the compact representation). The analysis then considers the effect of manipulating sets of parse tuples, rather than endpoints which are the values manipulated by recognizers. Parsers corresponding to grammars in CNF will return, in the worst case, for each start/end point pair (s, e) , $(((e - s) + 1) * t^2)$ parse tuples, where `t` is the number of terminals and non-terminals in the grammar. It follows that there are O(n) parse tuples for each parser and begin/endpoint pair. Each parse tuple corresponds to a bi-partition of the sequence starting at `s` and finishing at `e` by two parsers (possibly the same) from the set of parsers corresponding to terminals and non-terminals in the grammar. It is these parse tuples that are manipulated by `'orelse'` and `'thenS'`. The only effect on complexity of these operations is to increase the complexity of `'orelse'` from O(n) to O(n^2), which is the same as the complexity of `'thenS'`. Owing to the fact that the complexity of `'thenS'` had the highest degree in the application of a compound recognizer to an index, increasing the complexity of `'orelse'` to the same degree in parsing has no effect on the overall complexity of the process.

The representation of trees in the memotable has one entry for each parser. In the worst case, when the parser is applied to every index, the entry has `n` sub-entries, corresponding to `n` begin points. For each of these sub-entries there are up to `n` sub-sub-entries, each corresponding to an end point of the parse. Each of these sub-entries contains O(n) parse tuples as discussed above. It follows that the size of the compact representation is O(n^3).

7 Implementation

We have implemented our method in the pure functional programming language Haskell. We use a monad (Wadler 1995) to implement memoization. Use of a monad allows the memotable to be systematically threaded through the parsers while hiding the details of table update and reuse, allowing a clean and simple interface to be presented to the user. The complete Haskell code is available from any of the authors.

8 Experimental Results

In order to provide evidence of the low-order polynomial costs and scalability of our method, we conducted a limited evaluation with respect to four practical natural-language grammars used by Tomita (Appendix F, 1986) when comparing his algorithm with Earley's, and four variants of an abstract highly ambiguous grammar from Aho and Ullman (1972). Our Haskell program was compiled using the Glasgow Haskell Compiler 6.6 (the code has not yet been tuned to obtain the best performance from this pat-form). We used a 3GHz/1GB PC in our experiments.

8.1 Tomita's Grammars

The Tomita grammars used were: G1 (8 rules), G2 (40 rules), G3 (220 rules), and G4 (400 rules). We used two sets of input: a) the three most-ambiguous inputs from Tomita's sentence set 1 (Appendix G) of lengths `19, 26,` and `26` which we parsed with G3 (as did Tomita), and b) three inputs of lengths `4, 10,` and `40`, with systematically increasing ambiguity, chosen from Tomita's sentence set 2, which he generated automatically using the formula:

noun verb det noun (prep det noun)*

The results, which are tabulated in figure 1, show our timings and those recorded by Tomita for his original algorithm and for an improved Earley method, using a DEC-20 machine (Tomita 1986, Appendix D).

Considered by themselves our timings are low enough to suggest that our method is feasible for use in small to medium applications, such as NL interfaces to databases or rhythm analysis in poetry. Such applications typically have modest grammars (no more than a few hundred rules) and are not required to parse huge volumes of input.

Clearly there can be no direct comparison against years-old DEC-20 times, and improved versions of both of these algorithms do exist. However, we point to some relevant trends in the results. The increases in times for our method roughly mirror the increases shown for Tomita's algorithm, as grammar complexity and/or input size increase. This suggests that our algorithm scales adequately well, and not dissimilarly to the earlier algorithms.

8.2 Highly ambiguous abstract grammars

We defined four parsers as executable specifications of four variants of a highly-ambiguous grammar introduced by Aho and Ullman (1972) when discussing ambiguity: an unmemoized non-left–recursive parser `s`, a memoized version `ms`, a memoized left–recursive version `sml`, and a left–recursive version with all parts memoized. (This improves efficiency similarly to converting the grammar to Chomsky Normal Form.):

Input length	No. of Parses	Our algorithm (complete parsing)-PC				Tomitas (complete parsing)-DEC 20				Earleys (recognition only)-DEC 20			
		G1	G2	G3	G4	G1	G2	G3	G4	G1	G2	G3	G4
Input from Tomitas sentence set 1. Timings are in seconds.													
19	346			0.02				4.79				7.66	
26	1,464			0.03				8.66				14.65	
Input from Tomitas sentence set 2. Timings are in seconds.													
22	429	0.00	0.00	0.03	0.05	2.80	6.40	4.74	19.93	2.04	7.87	7.25	42.75
31	16,796	0.00	0.02	0.05	0.09	6.14	14.40	10.40	45.28	4.01	14.09	12.06	70.74
40	742,900	0.03	0.08	0.11	0.14	11.70	28.15	18.97	90.85	6.75	22.42	19.12	104.91

Figure 1: Informal comparison with Tomita/Earley results

```
s    = (term 'a' ‘thenS‘ s ‘thenS‘ s)
       ‘orelse‘ empty
sm   = memoize "sm"
       ((term 'a' ‘thenS‘ sm ‘thenS‘ sm)
       ‘orelse‘ empty)
sml  = memoize "sml"
       ((sml ‘thenS‘ sml
             ‘thenS‘ term 'a')
       ‘orelse‘ empty)
smml = memoize "smml"
       ((smml ‘thenS‘
         (memoize "smml_a"
           (smml ‘thenS‘ term 'a')))
       ‘orelse‘ empty)
```

We chose these four grammars as they are highly ambiguous. According to Aho and Ullman (1972), `s` generates over `128` billion complete parses of an input consisting of 24 `'a'`'s. Although the left-recursive grammar does not generate exactly the same parses, it generates the same number of parses, as it matches a terminal at the end of the rule rather than at the start.

Input length	No. of parses excluding partial parses	Seconds to generate the packed representation of full and partial parses			
		s	sm	sml	smml
6	132	1.22	0.00	0.00	0.00
12	208,012	out of space	0.00	0.00	0.02
24	1.29e+12		0.08	0.13	0.06
48	1.313e+26		0.83	0.97	0.80

Figure 2: Times to compute forest for n

These results show that our method can accommodate massively-ambiguous input involving the generation of large and complex parse forests. For example, the full forest for `n=48` contains 1,225 choice nodes and 19,600 branch nodes. Note also that the use of more memoization in `smml` reduces the cost of left-rec checking.

9 Concluding Comments

We have extended previous work of others on modular parsers constructed as executable specifications of grammars, in order to accommodate ambiguity and left recursion in polynomial time and space. We have implemented our method as a set of parser combinators in the functional programming language Haskell, and have conducted experiments which demonstrate the viability of the approach.

The results of the experiments suggest that our method is feasible for use in small to medium applications which need parsing of ambiguous grammars. Our method, like other methods which use parser combinators or DCGs, allows parsers to be created as executable specifications which are “embedded” in the host programming language. It is often claimed that this embedded approach is more convenient than indirect methods which involve the use of separate compiler tools such as yacc, for reasons such as support from the host language (including type checking) and ease of use. The major advantage of our method is that it increases the type of grammars that can be accommodated in the embedded style, by supporting left recursion and ambiguity. This greatly increases what can be done in this approach to parser construction, and removes the need for non-expert users to painfully rewrite and debug their grammars to avoid left recursion. We believe such advantages balance well against any reduction in performance, especially when an application is being prototyped.

The Haskell implementation is in its initial stage. We are in the process of modifying it to improve efficiency, and to make better use of Haskell’s lazy evaluation strategy (e.g. to return only the first `n` successful parses of the input).

Future work includes proof of correctness, analysis with respect to grammar size, testing with larger natural language grammars, and extending the approach so that language evaluators can be constructed as modular executable specifications of attribute grammars.

Acknowledgements

Richard Frost acknowledges the support provided by the Natural Sciences and Engineering Research Council of Canada in the form of a discovery grant.

References

1. Aho, A. V. and Ullman, J. D. (1972) *The Theory of Parsing, Translation, and Compiling. Volume I: Parsing.* Prentice-Hall.

2. Aho, A. V., Sethi, R. and Ullman, J. D. (1986) *Compilers: Principles, Techniques and Tools*. Addison-Wesley Longman Publishing Co.

3. Camarao, C., Figueiredo, L. and Oliveira, R.,H. (2003) Mimico: A Monadic Combinator Compiler Generator. *Journal of the Brazilian Computer Society* 9(1).

4. Earley, J. (1970) An efficient context-free parsing algorithm.*Comm. ACM* 13(2) 94–102.

5. Eijck, J. van (2003) Parser combinators for extraction. In Paul Dekker and Robert van Rooy, editors, *Proceedings of the Fourteenth Amsterdam Colloqium* ILLC, University of Amsterdam. 99–104.

6. Frost, R. A. (2006) Realization of Natural-Language Interfaces using Lazy Functional Programming. *ACM Comput. Surv.* 38(4).

7. Frost, R. A. and Hafiz, R. (2006) A New Top-Down Parsing Algorithm to Accommodate Ambiguity and Left Recursion in Polynomial Time. *SIGPLAN Notices* 42 (5) 46–54.

8. Hutton, G. (1992) Higher-order functions for parsing. *J. Functional Programming* 2 (3) 323–343.

9. Johnson, M. (1995) Squibs and Discussions: Memoization in top-down parsing. *Computational Linguistics* 21(3) 405–417.

10. Kay, M. (1980) Algorithm schemata and data structures in syntactic processing. *Technical Report CSL-80–12*, XEROX Palo Alto Research Center.

11. Koskimies, K. (1990) Lazy recursive descent parsing for modular language implementation. *Software Practice and Experience* 20 (8) 749–772.

12. Kuno, S. (1965) The predictive analyzer and a path elimination technique. *Comm. ACM* 8(7) 453–462.

13. Leermakers, R. (1993) *The Functional Treatment of Parsing*. Kluwer Academic Publishers, ISBN0–7923–9376–7.

14. Lickman, P. (1995) Parsing With Fixed Points. *Master's Thesis*, University of Cambridge.

15. Moore, R. C. (2000) Removing left recursion from context-free grammars. In *Proceedings, 1st Meeting of the North American Chapter of the Association for Computational Linguistics, Seattle, Washington, ANLP–NAACL 2000*. 249–255.

16. Nederhof, M. J. and Koster, C. H. A. (1993) Top-Down Parsing for Left-recursive Grammars. *TechnicalReport* 93–10 Research Institute for Declarative Systems, Department of Informatics, Faculty of Mathematics and Informatics, Katholieke Universiteit, Nijmegen.

17. Norvig, P. (1991) Techniques for automatic memoisation with applications to context-free parsing. *Computational Linguistics* 17(1) 91–98.

18. Shiel, B. A. (1976) Observations on context-free parsing. *Technical Report* TR 12–76, Center for Research in Computing Technology, Aiken Computational Laboratory, Harvard University.

19. Tomita, M. (1986) *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems.* Kluwer Academic Publishers, Boston, MA.

20. Wadler, P. (1995) Monads for functional programming, *Proceedings of the Baastad Spring School on Advanced Functional Programming,* ed J. Jeuring and E. Meijer. Springer Verlag LNCS 925.

On the Complexity of Non-Projective Data-Driven Dependency Parsing

Ryan McDonald
Google Inc.
76 Ninth Avenue
New York, NY 10028
ryanmcd@google.com

Giorgio Satta
University of Padua
via Gradenigo 6/A
I-35131 Padova, Italy
satta@dei.unipd.it

Abstract

In this paper we investigate several non-projective parsing algorithms for dependency parsing, providing novel polynomial time solutions under the assumption that each dependency decision is independent of all the others, called here the edge-factored model. We also investigate algorithms for non-projective parsing that account for non-local information, and present several hardness results. This suggests that it is unlikely that exact non-projective dependency parsing is tractable for any model richer than the edge-factored model.

1 Introduction

Dependency representations of natural language are a simple yet flexible mechanism for encoding words and their syntactic dependencies through directed graphs. These representations have been thoroughly studied in descriptive linguistics (Tesnière, 1959; Hudson, 1984; Sgall et al., 1986; Meĺčuk, 1988) and have been applied in numerous language processing tasks. Figure 1 gives an example dependency graph for the sentence *Mr. Tomash will remain as a director emeritus*, which has been extracted from the Penn Treebank (Marcus et al., 1993). Each edge in this graph represents a single syntactic dependency directed from a word to its modifier. In this representation all edges are labeled with the specific syntactic function of the dependency, e.g., SBJ for subject and NMOD for modifier of a noun. To simplify computation and some important definitions, an artificial token is inserted into the sentence as the left most word and will always represent the root of the dependency graph. We assume all dependency graphs are directed trees originating out of a single node, which is a common constraint (Nivre, 2005).

The dependency graph in Figure 1 is an example of a nested or *projective* graph. Under the assumption that the root of the graph is the left most word of the sentence, a projective graph is one where the edges can be drawn in the plane above the sentence with no two edges crossing. Conversely, a *non-projective* dependency graph does not satisfy this property. Figure 2 gives an example of a non-projective graph for a sentence that has also been extracted from the Penn Treebank. Non-projectivity arises due to long distance dependencies or in languages with flexible word order. For many languages, a significant portion of sentences require a non-projective dependency analysis (Buchholz et al., 2006). Thus, the ability to learn and infer non-projective dependency graphs is an important problem in multilingual language processing.

Syntactic dependency parsing has seen a number of new learning and inference algorithms which have raised state-of-the-art parsing accuracies for many languages. In this work we focus on *data-driven* models of dependency parsing. These models are not driven by any underlying grammar, but instead learn to predict dependency graphs based on a set of parameters learned solely from a labeled corpus. The advantage of these models is that they negate the need for the development of grammars when adapting the model to new languages.

One interesting class of data-driven models are

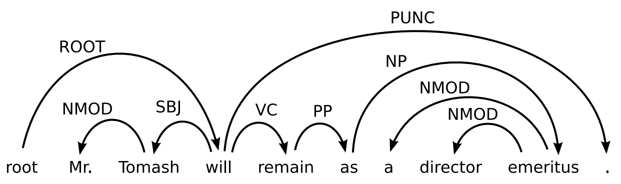


Figure 1: A projective dependency graph.

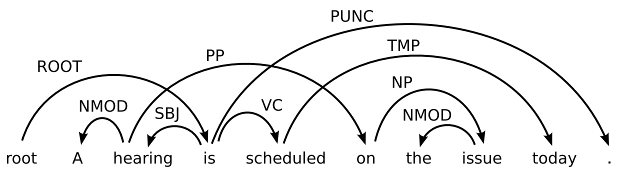


Figure 2: Non-projective dependency graph.

those that assume each dependency decision is independent modulo the global structural constraint that dependency graphs must be trees. Such models are commonly referred to as *edge-factored* since their parameters factor relative to individual edges of the graph (Paskin, 2001; McDonald et al., 2005a). Edge-factored models have many computational benefits, most notably that inference for non-projective dependency graphs can be achieved in polynomial time (McDonald et al., 2005b). The primary problem in treating each dependency as independent is that it is not a realistic assumption. Non-local information, such as arity (or valency) and neighbouring dependencies, can be crucial to obtaining high parsing accuracies (Klein and Manning, 2002; McDonald and Pereira, 2006). However, in the data-driven parsing setting this can be partially adverted by incorporating rich feature representations over the input (McDonald et al., 2005a).

The goal of this work is to further our current understanding of the computational nature of non-projective parsing algorithms for both learning and inference within the data-driven setting. We start by investigating and extending the edge-factored model of McDonald et al. (2005b). In particular, we appeal to the Matrix Tree Theorem for multi-digraphs to design polynomial-time algorithms for calculating both the partition function and edge expectations over all possible dependency graphs for a given sentence. To motivate these algorithms, we show that they can be used in many important learning and inference problems including min-risk decoding, training globally normalized log-linear models, syntactic language modeling, and unsupervised learning via the EM algorithm – none of which have previously been known to have exact non-projective implementations.

We then switch focus to models that account for non-local information, in particular arity and neighbouring parse decisions. For systems that model arity constraints we give a reduction from the Hamiltonian graph problem suggesting that the parsing problem is intractable in this case. For neighbouring parse decisions, we extend the work of McDonald and Pereira (2006) and show that modeling vertical neighbourhoods makes parsing intractable in addition to modeling horizontal neighbourhoods. A consequence of these results is that it is unlikely that exact non-projective dependency parsing is tractable for any model assumptions weaker than those made by the edge-factored models.

1.1 Related Work

There has been extensive work on data-driven dependency parsing for both projective parsing (Eisner, 1996; Paskin, 2001; Yamada and Matsumoto, 2003; Nivre and Scholz, 2004; McDonald et al., 2005a) and non-projective parsing systems (Nivre and Nilsson, 2005; Hall and Novák, 2005; McDonald et al., 2005b). These approaches can often be classified into two broad categories. In the first category are those methods that employ approximate inference, typically through the use of linear time shift-reduce parsing algorithms (Yamada and Matsumoto, 2003; Nivre and Scholz, 2004; Nivre and Nilsson, 2005). In the second category are those that employ exhaustive inference algorithms, usually by making strong independence assumptions, as is the case for edge-factored models (Paskin, 2001; McDonald et al., 2005a; McDonald et al., 2005b). Recently there have also been proposals for exhaustive methods that weaken the edge-factored assumption, including both approximate methods (McDonald and Pereira, 2006) and exact methods through integer linear programming (Riedel and Clarke, 2006) or branch-and-bound algorithms (Hirakawa, 2006).

For grammar based models there has been limited work on empirical systems for non-projective parsing systems, notable exceptions include the work of Wang and Harper (2004). Theoretical studies of note include the work of Neuhaus and Böker (1997) showing that the recognition problem for a minimal dependency grammar is hard. In addition, the work of Kahane et al. (1998) provides a polynomial parsing algorithm for a constrained class of non-projective structures. Non-projective dependency parsing can be related to certain parsing problems defined for phrase structure representations, as for instance immediate dominance CFG parsing (Barton et al., 1987) and shake-and-bake translation (Brew, 1992).

Independently of this work, Koo et al. (2007) and Smith and Smith (2007) showed that the Matrix-Tree Theorem can be used to train edge-factored log-linear models of dependency parsing. Both studies constructed implementations that compare favorably with the state-of-the-art. The work of Meilă and Jaakkola (2000) is also of note. In that study they use the Matrix Tree Theorem to develop a tractable bayesian learning algorithms for tree belief networks, which in many ways are closely related to probabilistic dependency parsing formalisms and the problems we address here.

2 Preliminaries

Let $L = \{l_1, \ldots, l_{|L|}\}$ be a set of permissible syntactic edge labels and $\boldsymbol{x} = x_0 x_1 \cdots x_n$ be a sentence such that x_0=root. From this sentence we construct a complete labeled directed graph (digraph) $G_{\boldsymbol{x}} = (V_{\boldsymbol{x}}, E_{\boldsymbol{x}})$ such that,

- $V_{\boldsymbol{x}} = \{0, 1, \ldots, n\}$
- $E_{\boldsymbol{x}} = \{(i,j)^k \mid \forall\, i,j \in V_{\boldsymbol{x}}$ and $1 \leq k \leq |L|\}$

$G_{\boldsymbol{x}}$ is a graph where each word in the sentence is a node, and there is a directed edge between every pair of nodes for every possible label. By its definition, $G_{\boldsymbol{x}}$ is a multi-digraph, which is a digraph that may have more than one edge between any two nodes. Let $(i,j)^k$ represent the k^{th} edge from i to j. $G_{\boldsymbol{x}}$ encodes all possible labeled dependencies between the words of $\boldsymbol{x}$. Thus every possible dependency graph of $\boldsymbol{x}$ must be a subgraph of $G_{\boldsymbol{x}}$.

Let $i \rightarrow^+ j$ be a relation that is true if and only if there is a non-empty directed path from node i to node j in some graph under consideration. A directed spanning tree[1] of a graph G, that originates out of node 0, is any subgraph $T = (V_T, E_T)$ such that,

- $V_T = V_{\boldsymbol{x}}$ and $E_T \subseteq E_{\boldsymbol{x}}$
- $\forall j \in V_T$, $0 \rightarrow^+ j$ if and only if $j \neq 0$
- If $(i,j)^k \in E_T$, then $(i',j)^{k'} \notin E_T$, $\forall i' \neq i$ and/or $k' \neq k$.

Define $T(G)$ as the set of all directed spanning trees for a graph G. As McDonald et al. (2005b) noted, there is a one-to-one correspondence between spanning trees of $G_{\boldsymbol{x}}$ and labeled dependency graphs of $\boldsymbol{x}$, i.e., $T(G_{\boldsymbol{x}})$ is exactly the set of all possible projective and non-projective dependency graphs for sentence $\boldsymbol{x}$. Throughout the rest of this paper, we will refer to any $T \in T(G_{\boldsymbol{x}})$ as a valid dependency graph for a sentence $\boldsymbol{x}$. Thus, by definition, every valid dependency graph must be a tree.

3 Edge-factored Models

In this section we examine the class of models that assume each dependency decision is independent. Within this setting, every edge in an induced graph $G_{\boldsymbol{x}}$ for a sentence $\boldsymbol{x}$ will have an associated weight $w^k_{ij} \geq 0$ that maps the k^{th} directed edge from node i to node j to a real valued numerical weight. These weights represents the likelihood of a dependency occurring from word w_i to word w_j with label l_k. Define the weight of a spanning tree $T = (V_T, E_T)$ as the product of the edge weights

$$w(T) = \prod_{(i,j)^k \in E_T} w^k_{ij}$$

It is easily shown that this formulation includes the projective model of Paskin (2001) and the non-projective model of McDonald et al. (2005b).

The definition of w^k_{ij} depends on the context in which it is being used. For example, in the work of McDonald et al. (2005b) it is simply a linear classifier that is a function of the words in the dependency, the label of the dependency, and any contextual features of the words in the sentence. In a generative probabilistic model (such as Paskin (2001)) it could represent the conditional probability of a word w_j being generated with a label l_k given that the word being modified is w_i (possibly with some other information such as the orientation of the dependency

[1] A directed spanning tree is commonly referred to as a arborescence in the graph theory literature.

or the number of words between w_i and w_j). We will attempt to make any assumptions about the form w_{ij}^k clear when necessary.

For the remainder of this section we discuss three crucial problems for learning and inference while showing that each can be computed tractably for the non-projective case.

3.1 Finding the Argmax

The first problem of interest is finding the highest weighted tree for a given input sentence $\boldsymbol{x}$

$$T = \operatorname*{argmax}_{T \in T(G_{\boldsymbol{x}})} \prod_{(i,j)^k \in E_T} w_{ij}^k$$

McDonald et al. (2005b) showed that this can be solved in $O(n^2)$ for unlabeled parsing using the Chu-Liu-Edmonds algorithm for standard digraphs (Chu and Liu, 1965; Edmonds, 1967). Unlike most exact projective parsing algorithms, which use efficient bottom-up chart parsing algorithms, the Chu-Liu-Edmonds algorithm is greedy in nature. It begins by selecting the single best incoming dependency edge for each node j. It then post-processes the resulting graph to eliminate cycles and then continues recursively until a spanning tree (or valid dependency graph) results (see McDonald et al. (2005b) for details).

The algorithm is trivially extended to the multi-digraph case for use in labeled dependency parsing. First we note that if the maximum directed spanning tree of a multi-digraph $G_{\boldsymbol{x}}$ contains any edge $(i,j)^k$, then we must have $k = k^* = \operatorname{argmax}_k w_{ij}^k$. Otherwise we could simply substitute $(i,j)^{k^*}$ in place of $(i,j)^k$ and obtain a higher weighted tree. Therefore, without effecting the solution to the argmax problem, we can delete all edges in $G_{\boldsymbol{x}}$ that do not satisfy this property. The resulting digraph is no longer a multi-digraph and the Chu-Liu-Edmonds algorithm can be applied directly. The new runtime is $O(|L|n^2)$.

As a side note, the k-best argmax problem for digraphs can be solved in $O(kn^2)$ (Camerini et al., 1980). This can also be easily extended to the multi-digraph case for labeled parsing.

3.2 Partition Function

A common step in many learning algorithms is to compute the sum over the weight of all the possible outputs for a given input $\boldsymbol{x}$. This value is often referred to as the *partition function* due to its similarity with a value by the same name in statistical mechanics. We denote this value as $Z_{\boldsymbol{x}}$,

$$Z_{\boldsymbol{x}} = \sum_{T \in T(G_{\boldsymbol{x}})} w(T) = \sum_{T \in T(G_{\boldsymbol{x}})} \prod_{(i,j)^k \in E_T} w_{i,j}^k$$

To compute this sum it is possible to use the Matrix Tree Theorem for multi-digraphs,

Matrix Tree Theorem (Tutte, 1984): *Let G be a multi-digraph with nodes $V = \{0, 1, \ldots, n\}$ and edges E. Define (Laplacian) matrix Q as a $(n+1)\times(n+1)$ matrix indexed from 0 to n. For all i and j, define:*

$$Q_{jj} = \sum_{i \neq j, (i,j)^k \in E} w_{ij}^k \quad \& \quad Q_{ij} = \sum_{i \neq j, (i,j)^k \in E} -w_{ij}^k$$

If the i^{th} row and column are removed from Q to produce the matrix Q^i, then the sum of the weights of all directed spanning trees rooted at node i is equal to $|Q^i|$ (the determinant of Q^i).

Thus, if we construct Q for a graph $G_{\boldsymbol{x}}$, then the determinant of the matrix Q^0 is equivalent to $Z_{\boldsymbol{x}}$. The determinant of an $n \times n$ matrix can be calculated in numerous ways, most of which take $O(n^3)$ (Cormen et al., 1990). The most efficient algorithms for calculating the determinant of a matrix use the fact that the problem is no harder than matrix multiplication (Cormen et al., 1990). Matrix multiplication currently has known $O(n^{2.38})$ implementations and it has been widely conjectured that it can be solved in $O(n^2)$ (Robinson, 2005). However, most algorithms with sub-$O(n^3)$ running times require constants that are large enough to negate any asymptotic advantage for the case of dependency parsing. As a result, in this work we use $O(n^3)$ as the runtime for computing $Z_{\boldsymbol{x}}$.

Since it takes $O(|L|n^2)$ to construct the matrix Q, the entire runtime to compute $Z_{\boldsymbol{x}}$ is $O(n^3 + |L|n^2)$.

3.3 Edge Expectations

Another important problem for various learning paradigms is to calculate the expected value of each edge for an input sentence $\boldsymbol{x}$,

$$\langle (i,j)^k \rangle_{\boldsymbol{x}} = \sum_{T \in T(G_{\boldsymbol{x}})} w(T) \times I((i,j)^k, T)$$

```
Input: $\boldsymbol{x} = x_0 x_1 \cdots x_n$
1.  Construct $Q$                                                  $O(|L|n^2)$
2.  for $j$ : 1 .. n                                               $O(n)$
3.      $Q'_{jj} = Q_{jj}$ and $Q'_{ij} = Q_{ij}, 0 \leq \forall i \leq n$     $O(n)$
4.      $Q_{jj} = 1$ and $Q_{ij} = 0, 0 \leq \forall i \leq n$            $O(n)$
5.      for $i$ : 0 .. n & $i \neq j$                              $O(n)$
6.          $Q_{ij} = -1$                                          $O(1)$
7.          $Z_{\boldsymbol{x}} = |Q^0|$                           $O(n^3)$
8.          $\langle (i,j)^k \rangle_{\boldsymbol{x}} = w^k_{ij} Z_{\boldsymbol{x}}, \forall 1 \leq k \leq |L|$     $O(|L|)$
9.      end for
10.     $Q_{jj} = Q'_{jj}$ and $Q_{ij} = Q'_{ij}, 0 \leq \forall i \leq n$     $O(n)$
11. end for
```

Figure 3: Algorithm to calculate $\langle (i,j)^k \rangle_{\boldsymbol{x}}$ in $O(n^5 + |L|n^2)$.

where $I((i,j)^k, T)$ is an indicator function that is one when the edge $(i,j)^k$ is in the tree T.

To calculate the expectation for the edge $(i,j)^k$, we can simply eliminate all edges $(i',j)^{k'} \neq (i,j)^k$ from $G_{\boldsymbol{x}}$ and calculate $Z_{\boldsymbol{x}}$. $Z_{\boldsymbol{x}}$ will now be equal to the sum of the weights of all trees that contain $(i,j)^k$. A naive implementation to compute the expectation of all $|L|n^2$ edges takes $O(|L|n^5 + |L|^2 n^4)$, since calculating $Z_{\boldsymbol{x}}$ takes $O(n^3 + |L|n^2)$ for a single edge. However, we can reduce this considerably by constructing Q a single time and only making modifications to it when necessary. An algorithm is given in Figure 3.3 that has a runtime of $O(n^5 + |L|n^2)$. This algorithm works by first constructing Q. It then considers edges from the node i to the node j. Now, assume that there is only a single edge from i to j and that that edge has a weight of 1. Furthermore assume that this edge is the only edge directed into the node j. In this case Q should be modified so that $Q_{jj} = 1$, $Q_{ij} = -1$, and $Q_{i'j} = 0$, $\forall i' \neq i, j$ (by the Matrix Tree Theorem). The value of $Z_{\boldsymbol{x}}$ under this new Q will be equivalent to the weight of all trees containing the single edge from i to j with a weight of 1. For a specific edge $(i,j)^k$ its expectation is simply $w^k_{ij} Z_{\boldsymbol{x}}$, since we can factor out the weight 1 edge from i to j in all the trees that contribute to $Z_{\boldsymbol{x}}$ and multiply through the actual weight for the edge. The algorithm then reconstructs Q and continues.

Following the work of Koo et al. (2007) and Smith and Smith (2007), it is possible to compute all expectations in $O(n^3 + |L|n^2)$ through matrix inversion. To make this paper self contained, we report here their algorithm adapted to our notation. First, consider the equivalence,

$$\begin{aligned}\frac{\partial \log Z_{\boldsymbol{x}}}{\partial w^k_{ij}} &= \frac{\partial \log Z_{\boldsymbol{x}}}{\partial Z_{\boldsymbol{x}}} \frac{\partial Z_{\boldsymbol{x}}}{\partial w^k_{ij}} \\ &= \frac{1}{Z_{\boldsymbol{x}}} \sum_{T \in T(G_{\boldsymbol{x}})} \frac{w(T)}{w^k_{ij}} \times I((i,j)^k, T)\end{aligned}$$

As a result, we can re-write the edge expectations as,

$$\langle (i,j)^k \rangle = Z_{\boldsymbol{x}} w^k_{ij} \frac{\partial \log Z_{\boldsymbol{x}}}{\partial w^k_{ij}} = Z_{\boldsymbol{x}} w^k_{ij} \frac{\partial \log |Q^0|}{\partial w^k_{ij}}$$

Using the chain rule, we get,

$$\frac{\partial \log |Q^0|}{\partial w^k_{ij}} = \sum_{i',j' \geq 1} \frac{\partial \log |Q^0|}{\partial (Q^0)_{i'j'}} \frac{\partial (Q^0)_{i'j'}}{\partial w^k_{ij}}$$

We assume the rows and columns of Q^0 are indexed from 1 so that the indexes of Q and Q^0 coincide. To calculate $\langle (i,j)^k \rangle$ when $i, j > 0$, we can use the fact that $\partial \log |X| / X_{ij} = (X^{-1})_{ji}$ and that $\partial (Q^0)_{i'j'} / \partial w^k_{ij}$ is non zero only when $i' = i$ and $j' = j$ or $i' = j' = j$ to get,

$$\langle (i,j)^k \rangle = Z_{\boldsymbol{x}} w^k_{ij} [((Q^0)^{-1})_{jj} - ((Q^0)^{-1})_{ji}]$$

When $i = 0$ and $j > 0$ the only non zero term of this sum is when $i' = j' = j$ and so

$$\langle (0,j)^k \rangle = Z_{\boldsymbol{x}} w^k_{0j} ((Q^0)^{-1})_{jj}$$

$Z_{\boldsymbol{x}}$ and $(Q^0)^{-1}$ can both be calculated a single time, each taking $O(n^3)$. Using these values, each expectation is computed in $O(1)$. Coupled with with the fact that we need to construct Q and compute the expectation for all $|L|n^2$ possible edges, in total it takes $O(n^3 + |L|n^2)$ time to compute all edge expectations.

3.4 Comparison with Projective Parsing

Projective dependency parsing algorithms are well understood due to their close connection to phrase-based chart parsing algorithms. The work of Eisner (1996) showed that the argmax problem for digraphs could be solved in $O(n^3)$ using a bottom-up dynamic programming algorithm similar to CKY. Paskin (2001) presented an $O(n^3)$ inside-outside algorithm for projective dependency parsing using the Eisner algorithm as its backbone. Using this algorithm it is trivial to calculate both $Z_{\boldsymbol{x}}$ and each

	Projective	Non-Projective
argmax	$O(n^3 + \|L\|n^2)$	$O(\|L\|n^2)$
$Z_{\boldsymbol{x}}$	$O(n^3 + \|L\|n^2)$	$O(n^3 + \|L\|n^2)$
$\langle (i,j)^k \rangle_{\boldsymbol{x}}$	$O(n^3 + \|L\|n^2)$	$O(n^3 + \|L\|n^2)$

Table 1: Comparison of runtime for non-projective and projective algorithms.

edge expectation. Crucially, the nested property of projective structures allows edge expectations to be computed in $O(n^3)$ from the inside-outside values. It is straight-forward to extend the algorithms of Eisner (1996) and Paskin (2001) to the labeled case adding only a factor of $O(|L|n^2)$.

Table 1 gives an overview of the computational complexity for the three problems considered here for both the projective and non-projective case. We see that the non-projective case compares favorably for all three problems.

4 Applications

To motivate the algorithms from Section 3, we present some important situations where each calculation is required.

4.1 Inference Based Learning

Many learning paradigms can be defined as inference-based learning. These include the perceptron (Collins, 2002) and its large-margin variants (Crammer and Singer, 2003; McDonald et al., 2005a). In these settings, a models parameters are iteratively updated based on the argmax calculation for a single or set of training instances under the current parameter settings. The work of McDonald et al. (2005b) showed that it is possible to learn a highly accurate non-projective dependency parser for multiple languages using the Chu-Liu-Edmonds algorithm for unlabeled parsing.

4.2 Non-Projective Min-Risk Decoding

In min-risk decoding the goal is to find the dependency graph for an input sentence $\boldsymbol{x}$, that on average has the lowest expected risk,

$$T = \operatorname*{argmin}_{T \in T(G_{\boldsymbol{x}})} \sum_{T' \in T(G_{\boldsymbol{x}})} w(T')\mathcal{R}(T,T')$$

where $\mathcal{R}$ is a risk function measuring the error between two graphs. Min-risk decoding has been studied for both phrase-structure parsing and dependency parsing (Titov and Henderson, 2006). In that work, as is common with many min-risk decoding schemes, $T(G_{\boldsymbol{x}})$ is not the entire space of parse structures. Instead, this set is usually restricted to a small number of possible trees that have been preselected by some baseline system. In this subsection we show that when the risk function is of a specific form, this restriction can be dropped. The result is an exact min-risk decoding procedure.

Let $\mathcal{R}(T,T')$ be the Hamming distance between two dependency graphs for an input sentence $\boldsymbol{x} = x_0 x_1 \cdots x_n$,

$$\mathcal{R}(T,T') = n \; - \sum_{(i,j)^k \in E_T} I((i,j)^k, T')$$

This is a common definition of risk between two graphs as it corresponds directly to labeled dependency parsing accuracy (McDonald et al., 2005a; Buchholz et al., 2006). Some algebra reveals,

$$\begin{aligned}
T &= \operatorname*{argmin}_{T \in T(G_{\boldsymbol{x}})} \sum_{T' \in T(G_{\boldsymbol{x}})} w(T')\mathcal{R}(T,T') \\
&= \operatorname*{argmin}_{T \in T(G_{\boldsymbol{x}})} \sum_{T' \in T(G_{\boldsymbol{x}})} w(T')[n \; - \sum_{(i,j)^k \in E_T} I((i,j)^k, T')] \\
&= \operatorname*{argmin}_{T \in T(G_{\boldsymbol{x}})} \; - \sum_{T' \in T(G_{\boldsymbol{x}})} w(T') \sum_{(i,j)^k \in E_T} I((i,j)^k, T') \\
&= \operatorname*{argmin}_{T \in T(G_{\boldsymbol{x}})} \; - \sum_{(i,j)^k \in E_T} \sum_{T' \in T(G_{\boldsymbol{x}})} w(T') I((i,j)^k, T') \\
&= \operatorname*{argmax}_{T \in T(G_{\boldsymbol{x}})} \sum_{(i,j)^k \in E_T} \sum_{T' \in T(G_{\boldsymbol{x}})} w(T') I((i,j)^k, T') \\
&= \operatorname*{argmax}_{T \in T(G_{\boldsymbol{x}})} \prod_{(i,j)^k \in E_T} e^{\sum_{T' \in T(G_{\boldsymbol{x}})} w(T') I((i,j)^k, T')} \\
&= \operatorname*{argmax}_{T \in T(G_{\boldsymbol{x}})} \prod_{(i,j)^k \in E_T} e^{\langle (i,j)^k \rangle_{\boldsymbol{x}}}
\end{aligned}$$

By setting the edge weights to $w_{ij}^k = e^{\langle (i,j)^k \rangle_{\boldsymbol{x}}}$ we can directly solve this problem using the edge expectation algorithm described in Section 3.3 and the argmax algorithm described in Section 3.1.

4.3 Non-Projective Log-Linear Models

Conditional Random Fields (CRFs) (Lafferty et al., 2001) are global discriminative learning algorithms for problems with structured output spaces, such as dependency parsing. For dependency parsing, CRFs would define the conditional probability of a dependency graph T for a sentence $\boldsymbol{x}$ as a globally normalized log-linear model,

$$
\begin{aligned}
p(T|\boldsymbol{x}) &= \frac{\prod_{(i,j)^k \in E_T} e^{\mathbf{w}\cdot\mathbf{f}(i,j,k)}}{\sum_{T' \in T(G_{\boldsymbol{x}})} \prod_{(i,j)^k \in E_{T'}} e^{\mathbf{w}\cdot\mathbf{f}(i,j,k)}} \\
&= \frac{\prod_{(i,j)^k \in E_T} w_{ij}^k}{\sum_{T' \in T(G_{\boldsymbol{x}})} \prod_{(i,j)^k \in E_{T'}} w_{ij}^k} \\
&= \frac{w(T)}{Z_{\boldsymbol{x}}}
\end{aligned}
$$

Here, the weights w_{ij}^k are potential functions over each edge defined as an exponentiated linear classifier with weight vector $\mathbf{w} \in \mathbb{R}^N$ and feature vector $\mathbf{f}(i,j,k) \in \mathbb{R}^N$, where $f_u(i,j,k) \in \mathbb{R}$ represents a single dimension of the vector $\mathbf{f}$. The denominator, which is exactly the sum over all graph weights, is a normalization constant forcing the conditional probability distribution to sum to one.

CRFs set the parameters $\mathbf{w}$ to maximize the log-likelihood of the conditional probability over a training set of examples $\mathcal{T} = \{(\boldsymbol{x}_\alpha, T_\alpha)\}_{\alpha=1}^{|\mathcal{T}|}$,

$$\mathbf{w} = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_\alpha \log\ p(T_\alpha|\boldsymbol{x}_\alpha)$$

This optimization can be solved through a variety of iterative gradient based techniques. Many of these require the calculation of feature expectations over the training set under model parameters for the previous iteration. First, we note that the feature functions factor over edges, i.e., $f_u(T) = \sum_{(i,j)^k \in E_T} f_u(i,j,k)$. Because of this, we can use edge expectations to compute the expectation of every feature f_u. Let $\langle f_u \rangle_{\boldsymbol{x}_\alpha}$ represent the expectation of feature f_u for the training instance $\boldsymbol{x}_\alpha$,

$$
\begin{aligned}
\langle f_u \rangle_{\boldsymbol{x}_\alpha} &= \sum_{T \in T(G_{\boldsymbol{x}_\alpha})} p(T|\boldsymbol{x}_\alpha) f_u(T) \\
&= \sum_{T \in T(G_{\boldsymbol{x}_\alpha})} p(T|\boldsymbol{x}_\alpha) \sum_{(i,j)^k \in E_T} f_u(i,j,k) \\
&= \sum_{T \in T(G_{\boldsymbol{x}_\alpha})} \frac{w(T)}{Z_{\boldsymbol{x}}} \sum_{(i,j)^k \in E_T} f_u(i,j,k) \\
&= \frac{1}{Z_{\boldsymbol{x}}} \sum_{(i,j)^k \in E_{\boldsymbol{x}_\alpha}} \sum_{T \in T(G_{\boldsymbol{x}})} w(T) I((i,j)^k, T) f_u(i,j,k) \\
&= \frac{1}{Z_{\boldsymbol{x}}} \sum_{(i,j)^k \in E_{\boldsymbol{x}_\alpha}} \langle (i,j)^k \rangle_{\boldsymbol{x}_\alpha} f_u(i,j,k)
\end{aligned}
$$

Thus, we can calculate the feature expectation per training instance using the algorithms for computing $Z_{\boldsymbol{x}}$ and edge expectations. Using this, we can calculate feature expectations over the entire training set,

$$\langle f_u \rangle_{\mathcal{T}} = \sum_\alpha p(\boldsymbol{x}_\alpha) \langle f_u \rangle_{\boldsymbol{x}_\alpha}$$

where $p(\boldsymbol{x}_\alpha)$ is typically set to $1/|\mathcal{T}|$.

4.4 Non-projective Generative Parsing Models

A generative probabilistic dependency model over some alphabet Σ consists of parameters $p_{x,y}^k$ associated with each dependency from word $x \in \Sigma$ to word $y \in \Sigma$ with label $l_k \in L$. In addition, we impose $0 \leq p_{x,y}^k \leq 1$ and the normalization conditions $\sum_{y,k} p_{x,y}^k = 1$ for each $x \in \Sigma$. We define a generative probability model p over trees $T \in T(G_{\boldsymbol{x}})$ and a sentence $\boldsymbol{x} = x_0 x_1 \cdots x_n$ conditioned on the sentence length, which is always known,

$$
\begin{aligned}
p(\boldsymbol{x}, T|n) &= p(\boldsymbol{x}|T,n) p(T|n) \\
&= \prod_{(i,j)^k \in E_T} p_{x_i,x_j}^k\ p(T|n)
\end{aligned}
$$

We assume that $p(T|n) = \beta$ is uniform. This model is studied specifically by Paskin (2001). In this model, one can view the sentence as being generated recursively in a top-down process. First, a tree is generated from the distribution $p(T|n)$. Then starting at the root of the tree, every word generates all of its modifiers independently in a recursive breadth-first manner. Thus, $p_{x,y}^k$ represents the probability of the word x generating its modifier y with label l_k. This distribution is usually smoothed and is often conditioned on more information including the orientation of x relative to y (i.e., to the left/right) and distance between the two words. In the supervised setting this model can be trained with maximum likelihood estimation, which amounts to simple counts over the data. Learning in the unsupervised setting requires EM and is discussed in Section 4.4.2.

Another generative dependency model of interest is that given by Klein and Manning (2004). In this model the sentence and tree are generated jointly, which allows one to drop the assumption that $p(T|n)$ is uniform. This requires the addition to the model of parameters $p_{x,\text{STOP}}$ for each $x \in \Sigma$, with the normalization condition $p_{x,\text{STOP}} + \sum_{y,k} p_{x,y}^k = 1$. It is possible to extend the model of Klein and Manning

(2004) to the non-projective case. However, the resulting distribution will be over multisets of words from the alphabet instead of strings. The discussion in this section is stated for the model in Paskin (2001); a similar treatment can be developed for the model in Klein and Manning (2004).

4.4.1 Language Modeling

A generative model of dependency structure might be used to determine the probability of a sentence $\boldsymbol{x}$ by marginalizing out all possible dependency trees,

$$\begin{aligned} p(\boldsymbol{x}|n) &= \sum_{T \in T(G_{\boldsymbol{x}})} p(\boldsymbol{x}, T|n) \\ &= \sum_{T \in T(G_{\boldsymbol{x}})} p(\boldsymbol{x}|T, n) p(T|n) \\ &= \beta \sum_{T \in T(G_{\boldsymbol{x}})} \prod_{(i,j)^k \in E_T} p^k_{x_i,x_j} = \beta Z_{\boldsymbol{x}} \end{aligned}$$

This probability can be used directly as a non-projective syntactic language model (Chelba et al., 1997) or possibly interpolated with a separate n-gram model.

4.4.2 Unsupervised Learning

In unsupervised learning we train our model on a sample of unannotated sentences $\mathcal{X} = \{\boldsymbol{x}_\alpha\}_{\alpha=1}^{|\mathcal{X}|}$. Let $|\boldsymbol{x}_\alpha| = n_\alpha$ and $p(T|n_\alpha) = \beta_\alpha$. We choose the parameters that maximize the log-likelihood

$$\begin{aligned} &\sum_{\alpha=1}^{|\mathcal{X}|} \log(p(\boldsymbol{x}_\alpha|n_\alpha)) = \\ &= \sum_{\alpha=1}^{|\mathcal{X}|} \log(\sum_{T \in T(G_{\boldsymbol{x}_\alpha})} p(\boldsymbol{x}_\alpha|T, n_\alpha)) + \sum_{\alpha=1}^{|\mathcal{X}|} \log(\beta_\alpha), \end{aligned}$$

viewed as a function of the parameters and subject to the normalization conditions, i.e., $\sum_{y,k} p^k_{x,y} = 1$ and $p^k_{x,y} \geq 0$.

Let $x_{\alpha i}$ be the i^{th} word of $\boldsymbol{x}_\alpha$. By solving the above constrained optimization problem with the usual Lagrange multipliers method one gets

$$\begin{aligned} &p^k_{x,y} = \\ &= \frac{\sum_{\alpha=1}^{|\mathcal{X}|} \frac{1}{Z_{\boldsymbol{x}_\alpha}} \sum_{\substack{i\,:\,x_{\alpha i} = x, \\ j\,:\,x_{\alpha j} = y}} \langle (i,j)^k \rangle_{\boldsymbol{x}_\alpha}}{\sum_{\alpha=1}^{|\mathcal{X}|} \frac{1}{Z_{\boldsymbol{x}_\alpha}} \sum_{y',k'} \sum_{\substack{i\,:\,x_{\alpha i} = x, \\ j'\,:\,x_{\alpha j'} = y'}} \langle (i,j')^{k'} \rangle_{\boldsymbol{x}_\alpha}}, \end{aligned}$$

where for each $\boldsymbol{x}_\alpha$ the expectation $\langle (i,j)^k \rangle_{\boldsymbol{x}_\alpha}$ is defined as in Section 3, but with the weight $w(T)$ replaced by the probability distribution $p(\boldsymbol{x}_\alpha|T, n_\alpha)$.

The above $|L| \cdot |\Sigma|^2$ relations represent a non-linear system of equations. There is no closed form solution in the general case, and one adopts the expectation maximization (EM) method, which is a specialization of the standard fixed-point iteration method for the solution of non-linear systems. We start with some initial assignment of the parameters and at each iteration we use the induced distribution $p(\boldsymbol{x}_\alpha|T, n_\alpha)$ to compute a refined value for the parameters themselves. We are always guaranteed that the Kullback-Liebler divergence between two approximated distributions computed at successive iterations does not increase, which implies the convergence of the method to some local maxima (with the exception of saddle points).

Observe that at each iteration we can compute quantities $\langle (i,j)^k \rangle_{\boldsymbol{x}_\alpha}$ and $Z_{\boldsymbol{x}_\alpha}$ in polynomial time using the algorithms from Section 3 with $p^k_{x_{\alpha i},x_{\alpha j}}$ in place of $w^k_{i,j}$. Furthermore, under some standard conditions the fixed-point iteration method guarantees a constant number of bits of precision gain for the parameters at each iteration, resulting in overall polynomial time computation in the size of the input and in the required number of bits for the precision. As far as we know, this is the first EM learning algorithm for the model in Paskin (2001) working in the non-projective case. The projective case has been investigated in Paskin (2001).

5 Beyond Edge-factored Models

We have shown that several computational problems related to parsing can be solved in polynomial time for the class of non-projective dependency models with the assumption that dependency relations are mutually independent. These independence assumptions are unwarranted, as it has already been established that modeling non-local information such as arity and nearby parsing decisions improves the accuracy of dependency models (Klein and Manning, 2002; McDonald and Pereira, 2006).

In the spirit of our effort to understand the nature of exact non-projective algorithms, we examine dependency models that introduce arity constraints as well as permit edge decisions to be dependent on a limited neighbourhood of other edges in the graph. Both kinds of models can no longer be considered edge-factored, since the likelihood of a dependency occurring in a particular analysis is now dependent on properties beyond the edge itself.

5.1 Arity

One feature of the edge-factored models is that no restriction is imposed on the arity of the nodes in the dependency trees. As a consequence, these models can generate dependency trees of unbounded arity. We show below that this is a crucial feature in the development of the complexity results we have obtained in the previous sections.

Let us assume a graph $G_{\boldsymbol{x}}^{(\phi)} = (V_{\boldsymbol{x}}, E_{\boldsymbol{x}})$ defined as before, but with the additional condition that each node $i \in V_{\boldsymbol{x}}$ is associated with an integer value $\phi(i) \geq 0$. $T(G_{\boldsymbol{x}}^{(\phi)})$ is now defined as the set of all directed spanning trees for $G_{\boldsymbol{x}}^{(\phi)}$ rooted in node 0, such that every node $i \in V_{\boldsymbol{x}}$ has arity smaller than or equal to $\phi(i)$. We now introduce a construction that will be used to establish several hardness results for the computational problems discussed in this paper. Recall that a Hamiltonian path in a directed graph G is a directed path that visits all of the nodes of G exactly once.

Let G be some directed graph with set of nodes $V = \{1, 2, \ldots, n\}$. We construct a target graph $G_{\boldsymbol{x}}^{(\phi)} = (V_{\boldsymbol{x}}, E_{\boldsymbol{x}})$ with $V_{\boldsymbol{x}} = V \cup \{0\}$ (0 the root node) and $|L| = 1$. For each $i, j \in V_{\boldsymbol{x}}$ with $i \neq j$, we add an edge $(i, j)^1$ to $E_{\boldsymbol{x}}$. We set $w_{i,j}^1 = 1$ if there is an edge from i to j in G, or else if i or j is the root node 0, and $w_{i,j}^1 = 0$ otherwise. Furthermore, we set $\phi(i) = 1$ for each $i \in V_{\boldsymbol{x}}$. This construction can be clearly carried out in log-space.

Note that each $T \in T(G_{\boldsymbol{x}}^{(\phi)})$ must be a monadic tree with weight equal to either 0 or 1. It is not difficult to see that if $w(T) = 1$, then when we remove the root node 0 from T we obtain a Hamiltonian path in G. Conversely, each Hamiltonian path in G can be extended to a spanning tree $T \in T(G_{\boldsymbol{x}}^{(\phi)})$ with $w(T) = 1$, by adding the root node 0.

Using the above observations, it can be shown that the solution of the argmax problem for $G_{\boldsymbol{x}}^{(\phi)}$ provides some Hamiltonian directed path in G. The latter search problem is FNP-hard, and is unlikely to be solved in polynomial time. Furthermore, quantity $Z_{\boldsymbol{x}}$ provides the count of the Hamiltonian directed paths in G, and for each $i \in V$, the expectation $\langle (0, i)^1 \rangle_{\boldsymbol{x}}$ provides the count of the Hamiltonian directed paths in G starting from node i. Both these counting problems are #P-hard, and very unlikely to have polynomial time solutions.

This result helps to relate the hardness of data-driven models to the commonly known hardness results in the grammar-driven literature given by Neuhaus and Böker (1997). In that work, an arity constraint is included in their minimal grammar.

5.2 Vertical and Horizontal Markovization

In general, we would like to say that every dependency decision is dependent on every other edge in a graph. However, modeling dependency parsing in such a manner would be a computational nightmare. Instead, we would like to make a Markov assumption over the edges of the tree, in a similar way that a Markov assumption can be made for sequential classification problems in order to ensure tractable learning and inference.

Klein and Manning (2003) distinguish between two kinds of Markovization for unlexicalized CFG parsing. The first is vertical Markovization, which makes the generation of a non-terminal dependent on other non-terminals that have been generated at different levels in the phrase-structure tree. The second is horizontal Markovization, which makes the generation of a non-terminal dependent on other non-terminals that have been generated at the same level in the tree.

For dependency parsing there are analogous notions of vertical and horizontal Markovization for a given edge $(i, j)^k$. First, let us define the vertical and horizontal *neighbourhoods* of $(i, j)^k$. The vertical neighbourhood includes all edges in any path from the root to a leaf that passes through $(i, j)^k$. The horizontal neighbourhood contains all edges $(i, j')^{k'}$. Figure 4 graphically displays the vertical and horizontal neighbourhoods for an edge in the dependency graph from Figure 1.

Vertical and horizontal Markovization essentially allow the score of the graph to factor over a larger scope of edges, provided those edges are in the same vertical or horizontal neighbourhood. A d^{th} order factorization is one in which the score factors only over the d nearest edges in the neighbourhoods. In

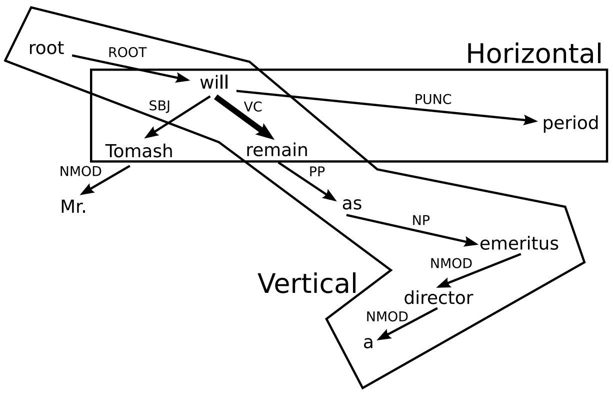


Figure 4: Vertical and Horizontal neighbourhood for the edge from *will* to *remain*.

McDonald and Pereira (2006), it was shown that non-projective dependency parsing with horizontal Markovization is FNP-hard. In this study we complete the picture and show that vertical Markovization is also FNP-hard.

Consider a first-order vertical Markovization in which the score for a dependency graph factors over pairs of vertically adjacent edges[2],

$$w(T) = \prod_{(h,i)^k,(i,j)^{k'} \in E_T} {}^{k}_{hi}w^{k'}_{ij}$$

where ${}^{k}_{hi}w^{k'}_{ij}$ is the weight of including both edges $(h,i)^k$ and $(i,j)^{k'}$ in the dependency graph. Note that this formulation does not include any contributions from dependencies that have no vertically adjacent neighbours, i.e., any edge $(0,i)^k$ such that there is no edge $(i,j)^{k'}$ in the graph. We can easily rectify this by inserting a second root node, say $0'$, and including the weights ${}^{k}_{0'0}w^{k'}_{0i}$. To ensure that only valid dependency graphs get a weight greater than zero, we can set ${}^{k}_{hi}w^{k'}_{ij} = 0$ if $i = 0'$ and ${}^{k}_{0'i}w^{k'}_{ij} = 0$ if $i \neq 0$.

Now, consider the NP-complete 3D-matching problem (3DM). As input we are given three sets of size m, call them A, B and C, and a set $S \subseteq A \times B \times C$. The 3DM problem asks if there is a set $S' \subseteq S$ such that $|S'| = m$ and for any two tuples $(a,b,c),(a',b',c') \in S'$ it is the case that $a \neq a'$, $b \neq b'$, and $c \neq c'$.

[2]McDonald and Pereira (2006) define this as a second-order Markov assumption. This is simply a difference in terminology and does not represent any meaningful distinction.

We can reduce the 3D-matching problem to the first-order vertical Markov parsing problem by constructing a graph $G = (V,E)$, such that $L = A \cup B \cup C$, $V = \{0', 0\} \cup A \cup B \cup C$ and $E = \{(i,j)^k \mid i,j \in V,\ k \in L\}$. The set E contains multiple edges between ever pair of nodes, each edge taking on a label representing a single element of the set $A \cup B \cup C$. Now, define ${}^{k}_{0'0}w^{k'}_{0a} = 1$, for all $a \in A$ and $k, k' \in A \cup B \cup C$, and ${}^{b}_{0a}w^{c}_{ab} = 1$, for all $a \in A$ and $b \in B$ and $c \in C$, and ${}^{c}_{ab}w^{c}_{bc} = 1$, for all $(a,b,c) \in S$. All other weights are set to zero.

We show below that there exists a bijection between the set of valid 3DMs for S and the set of non-zero weighted dependency graphs in $T(G)$. First, it is easy to show that for any 3DM S', there is a representative dependency graph that has a weight of 1. This graph simply consists of the edges $(0,a)^b$, $(a,b)^c$, and $(b,c)^c$, for all $(a,b,c) \in S'$, plus an arbitrarily labeled edge from $0'$ to 0.

To prove the reverse, consider a graph with weight 1. This graph must have a weight 1 edge into the node a of the form $(0,a)^b$ since the graph must be spanning. By the definition of the weight function, in any non-zero weighted tree, a must have a single outgoing edge, and that edge must be directed into the node b. Let's say that this edge is $(a,b)^c$. Then again by the weighting function, in any non-zero weighted graph, b must have a single outgoing edge that is directed into c, in particular the edge $(b,c)^c$. Thus, for any node a, there is a single path directed out of it to a single leaf $c \in C$. We can then state that the only non-zero weighted dependency graph is one where each $a \in A$, $b \in B$ and $c \in C$ occurs in exactly one of m disjoint paths from the root of the form $0 \to a \to b \to c$. This is because the label of the single edge going into node a will determine exactly the node b that the one outgoing edge from a must go into. The label of that edge determines exactly the single outgoing edge from b into some node c. Now, since the weighting function ensures that the only non-zero weighted paths into any leaf node c correspond directly to elements of S, each of the m disjoint paths represent a single tuple in a 3DM. Thus, if there is a non-zero weighted graph in $T(G)$, then it must directly correspond to a valid 3DM, which concludes the proof.

Note that any d^{th} order Markovization can be embedded into a $d+1^{th}$ Markovization. Thus, this result also holds for any arbitrary Markovization.

6 Discussion

In this paper we have shown that many important learning and inference problems can be solved efficiently for non-projective edge-factored dependency models by appealing to the Matrix Tree Theorem for multi-digraphs. These results extend the work of McDonald et al. (2005b) and help to further our understanding of when exact non-projective algorithms can be employed. When this analysis is coupled with the projective parsing algorithms of Eisner (1996) and Paskin (2001) we begin to get a clear picture of the complexity for data-driven dependency parsing within an edge-factored framework. To further justify the algorithms presented here, we outlined a few novel learning and inference settings in which they are required.

However, for the non-projective case, moving beyond edge-factored models will almost certainly lead to intractable parsing problems. We have provided further evidence for this by proving the hardness of incorporating arity constraints and horizontal/vertical edge Markovization, both of which incorporate information unavailable to an edge-factored model. The hardness results provided here are also of interest since both arity constraints and Markovization can be incorporated efficiently in the projective case through the straight-forward augmentation of the underlying chart parsing algorithms used in the projective edge-factored models. This highlights a fundamental difference between the nature of projective parsing algorithms and non-projective parsing algorithms. On the projective side, all algorithms use a bottom-up chart parsing framework to search the space of nested constructions. On the non-projective side, algorithms are either greedy-recursive in nature (i.e., the Chu-Liu-Edmonds algorithm) or based on the calculation of the determinant of a matrix (i.e., the partition function and edge expectations).

Thus, the existence of bottom-up chart parsing algorithms for projective dependency parsing provides many advantages. As mentioned above, it permits simple augmentation techniques to incorporate non-local information such as arity constraints and Markovization. It also ensures the compatibility of projective parsing algorithms with many important natural language processing methods that work within a bottom-up chart parsing framework, including information extraction (Miller et al., 2000) and syntax-based machine translation (Wu, 1996).

The complexity results given here suggest that polynomial chart-parsing algorithms do not exist for the non-projective case. Otherwise we should be able to augment them and move beyond edge-factored models without encountering intractability – just like the projective case. An interesting line of research is to investigate classes of non-projective structures that can be parsed with chart-parsing algorithms and how these classes relate to the languages parsable by other syntactic formalisms.

Acknowledgments

Thanks to Ben Taskar for pointing out the work of Meilă and Jaakkola (2000). Thanks to David Smith, Noah Smith and Michael Collins for making drafts of their EMNLP papers available.

References

G. E. Barton, R. C. Berwick, and E. S. Ristad. 1987. *Computational Complexity and Natural Language*. MIT Press, Cambridge, MA.

C. Brew. 1992. Letting the cat out of the bag: Generation for Shake-and-Bake MT. In *Proc. COLING*.

S. Buchholz, E. Marsi, A. Dubey, and Y. Krymolowski. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proc. CoNLL*.

P. M. Camerini, L. Fratta, and F. Maffioli. 1980. The k best spanning arborescences of a network. *Networks*, 10(2):91–110.

C. Chelba, D. Engle, F. Jelinek, V. Jimenez, S. Khudanpur, L. Mangu, H. Printz, E.S. Ristad, R. Rosenfeld, A. Stolcke, and D. Wu. 1997. Structure and performance of a dependency language model. In *Eurospeech*.

Y.J. Chu and T.H. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

M. Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proc. EMNLP*.

T.H. Cormen, C.E. Leiserson, and R.L. Rivest. 1990. *Introduction to Algorithms*. MIT Press/McGraw-Hill.

K. Crammer and Y. Singer. 2003. Ultraconservative online algorithms for multiclass problems. *JMLR*.

J. Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.

J. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proc. COLING*.

K. Hall and V. Nóvák. 2005. Corrective modeling for non-projective dependency parsing. In *Proc. IWPT*.

H. Hirakawa. 2006. Graph branch algorithm: An optimum tree search method for scored dependency graph with arc co-occurrence constraints. In *Proc. ACL*.

R. Hudson. 1984. *Word Grammar*. Blackwell.

S. Kahane, A. Nasr, and O Rambow. 1998. Pseudo-projectivity: A polynomially parsable non-projective dependency grammar. In *Proc. ACL*.

D. Klein and C.D. Manning. 2002. Fast exact natural language parsing with a factored model. In *Proc. NIPS*.

D. Klein and C. Manning. 2003. Accurate unlexicalized parsing. In *Proc. ACL*.

D. Klein and C. Manning. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proc. ACL*.

T. Koo, A. Globerson, X. Carreras, and M. Collins. 2007. Structured prediction models via the matrix-tree theorem. In *Proc. EMNLP*.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML*.

M. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

R. McDonald and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proc EACL*.

R. McDonald, K. Crammer, and F. Pereira. 2005a. Online large-margin training of dependency parsers. In *Proc. ACL*.

R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proc. HLT/EMNLP*.

M. Meilă and T. Jaakkola. 2000. Tractable Bayesian learning of tree belief networks. In *Proc. UAI*.

I.A. Meĺčuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press.

S. Miller, H. Fox, L.A. Ramshaw, and R.M. Weischedel. 2000. A novel use of statistical parsing to extract information from text. In *Proc NAACL*, pages 226–233.

P. Neuhaus and N. Böker. 1997. The complexity of recognition of linguistically adequate dependency grammars. In *Proc. ACL*.

J. Nivre and J. Nilsson. 2005. Pseudo-projective dependency parsing. In *Proc. ACL*.

J. Nivre and M. Scholz. 2004. Deterministic dependency parsing of english text. In *Proc. COLING*.

J. Nivre. 2005. Dependency grammar and dependency parsing. Technical Report MSI report 05133, Växjö University: School of Mathematics and Systems Engineering.

M.A. Paskin. 2001. Cubic-time parsing and learning algorithms for grammatical bigram models. Technical Report UCB/CSD-01-1148, Computer Science Division, University of California Berkeley.

S. Riedel and J. Clarke. 2006. Incremental integer linear programming for non-projective dependency parsing. In *Proc. EMNLP*.

S. Robinson. 2005. Toward an optimal algorithm for matrix multiplication. *News Journal of the Society for Industrial and Applied Mathematics*, 38(9).

P. Sgall, E. Hajičová, and J. Panevová. 1986. *The Meaning of the Sentence in Its Pragmatic Aspects*. Reidel.

D.A. Smith and N.A. Smith. 2007. Probabilistic models of nonprojective dependency trees. In *Proc. EMNLP*.

L. Tesnière. 1959. *Éléments de syntaxe structurale*. Editions Klincksieck.

I. Titov and J. Henderson. 2006. Bayes risk minimization in natural language parsing. University of Geneva technical report.

W.T. Tutte. 1984. *Graph Theory*. Cambridge University Press.

W. Wang and M. P. Harper. 2004. A statistical constraint dependency grammar (CDG) parser. In *Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL)*.

D. Wu. 1996. A polynomial-time algorithm for statistical machine translation. In *Proc. ACL*.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proc. IWPT*.

Dependency Parsing with Second-Order Feature Maps and Annotated Semantic Information

Massimiliano Ciaramita
Yahoo! Research
Ocata 1, S-08003
Barcelona, Spain
massi@yahoo-inc.com

Giuseppe Attardi
Dipartimento di Informatica
Università di Pisa
L. B. Pontecorvo 3, I-56127
Pisa, Italy
attardi@di.unipi.it

Abstract

This paper investigates new design options for the feature space of a dependency parser. We focus on one of the simplest and most efficient architectures, based on a deterministic shift-reduce algorithm, trained with the perceptron. By adopting second-order feature maps, the primal form of the perceptron produces models with comparable accuracy to more complex architectures, with no need for approximations. Further gains in accuracy are obtained by designing features for parsing extracted from semantic annotations generated by a tagger. We provide experimental evaluations on the Penn Treebank.

1 Introduction

A dependency tree represents a sentence as a labeled directed graph encoding syntactic and semantic information. The labels on the arcs can represent basic grammatical relations such as “subject” and “object”. Dependency trees capture grammatical structures that can be useful in several language processing tasks such as information extraction (Culotta & Sorensen, 2004) and machine translation (Ding & Palmer, 2005). Dependency treebanks are becoming available in many languages, and several approaches to dependency parsing on multiple languages have been evaluated in the CoNLL 2006 and 2007 shared tasks (Buchholz & Marsi, 2006; Nivre et al., 2007).

Dependency parsing is simpler than constituency parsing, since dependency trees do not have extra non-terminal nodes and there is no need for a grammar to generate them. Approaches to dependency parsing either generate such trees by considering all possible spanning trees (McDonald et al., 2005), or build a single tree by means of shift-reduce parsing actions (Yamada & Matsumoto, 2003). Deterministic dependency parsers which run in linear time have also been developed (Nivre & Scholz, 2004; Attardi, 2006). These parsers process the sentence sequentially, hence their efficiency makes them suitable for processing large amounts of text, as required, for example, in information retrieval applications.

Recent work on dependency parsing has highlighted the benefits of using rich feature sets and high-order modeling. Yamada and Matsumoto (2003) showed that learning an SVM model in the dual space with higher-degree polynomial kernel functions improves significantly the parser’s accuracy. McDonald and Pereira (2006) have shown that incorporating second order features relating to adjacent edge pairs improves the accuracy of maximum spanning tree parsers (MST). In the SVM-based approach, if the training data is large, it is not feasible to train a single model. Rather, Yamada and Matsumoto (see also (Hall et al., 2006)) partition the training data in different sets, on the basis of Part-of-Speech, then train one dual SVM model per set. While this approach simplifies the learning task it makes the parser more sensitive to the error rate of the POS tagger. The second-order MST algorithm has cubic time complexity. For non-projective languages the algorithm is NP-hard and McDonald and Pereira (2006) introduce an approximate algorithm to handle such cases.

In this paper we extend shift reduce parsing with second-order feature maps which explicitly represent all feature pairs. Also the augmented feature sets impose additional computational costs. However, excellent efficiency/accuracy trade-off is achieved by using the perceptron algorithm, without the need to resort to approximations, producing high-accuracy classifiers based on a single model.

We also evaluate a novel set of features for parsing. Recently various forms of shallow semantic processing have been investigated such as named-entity recognition (NER), semantic role labeling (SRL) and relation extraction. Syntactic parsing can provide useful features for these tasks; e.g., Punyakanok et al. (2005) show that full parsing is effective for semantic role labeling (see also related approaches evaluated within the CoNNL 2005 shared task (Carreras et al., 2005)). However, no evidence has been provided so far that annotated semantic information can be leveraged for improving parser performance. We report experiments showing that adding features extracted by an entity tagger improves the accuracy of a dependency parser.

2 Dependency parsing

A dependency parser takes as input a sentence s and returns a dependency graph d. Figure 1 shows a dependency tree for the sentence "Last week CBS Inc. canceled 'The People Next Door'."[1]. Dependencies are represented as labeled arrows from the *head* of the relation to the *modifier* word; thus, in the example, "Inc." is the modifier of a dependency labeled "SUB" (subject) to the main verb, the head, "canceled".

In statistical syntactic parsing a generator (e.g., a PCFG) is used to produce a number of candidate trees (Collins, 2000) with associated probability scores. This approach has been used also for dependency parsing, generating spanning trees as candidates and computing the maximum spanning tree (MST) using discriminative learning algorithms (McDonald et al., 2005). Second-order MST dependency parsers currently represent the state of the art in terms of accuracy. Yamada and Matsumoto (2003) proposed a deterministic classifier-based parser. Instead of learning directly which tree to assign to a sentence, the parser learns which *Shift/Reduce* actions to use in building the tree. Parsing is cast as a classification problem: at each step the parser applies a classifier to the features representing its current state to predict which action to perform on the tree. Similar deterministic approaches to parsing have been investigated also in the context of constituent parsing (Wong & Wu, 1999; Kalt, 2004).

[1]The figure also contains entity annotations which will be explained below in Section 4.1.

Nivre and Scholz (2004) proposed a variant of the model of Yamada and Matsumoto that reduces the complexity, from the worst case quadratic to linear. Attardi (2006) proposed a variant of the rules that handle non-projective relations while parsing deterministically in a single pass. Shift-reduce algorithms are simple and efficient, yet competitive in terms of accuracy: in the CoNLL-X shared task, for several languages, there was no statistically significant difference between second-order MST parsers and shift-reduce parsers.

3 A shift-reduce parser

We build upon DeSR, the shift-reduce parser described in (Attardi, 2006). This and Nivre and Scholz's (2004) provide among the simplest and most efficient methods. This parser constructs dependency trees by scanning input sentences in a single left-to-right pass and performing shift/reduce parsing actions. The parsing algorithm is fully deterministic and has linear complexity. The parser's behavior can be described as repeatedly selecting and applying a parsing rule to transform its state, while advancing through the sentence. Each token is analyzed once and a decision is made locally concerning the action to take, that is, without considering global properties of the tree being built. Nivre (2004) investigated the issue of (strict) incrementality for this type of parsers; i.e., if at any point of the analysis the processed input forms one connected structure. Nivre found that strict incrementality is not guaranteed within this parsing framework, although for correctly parsed trees the property holds in almost 90% of the cases.

3.1 Parsing algorithm

The state of the parser is represented by a triple $\langle S, I, A \rangle$, where S is the stack, I is the list of input tokens that remain to be processed and A is the arc

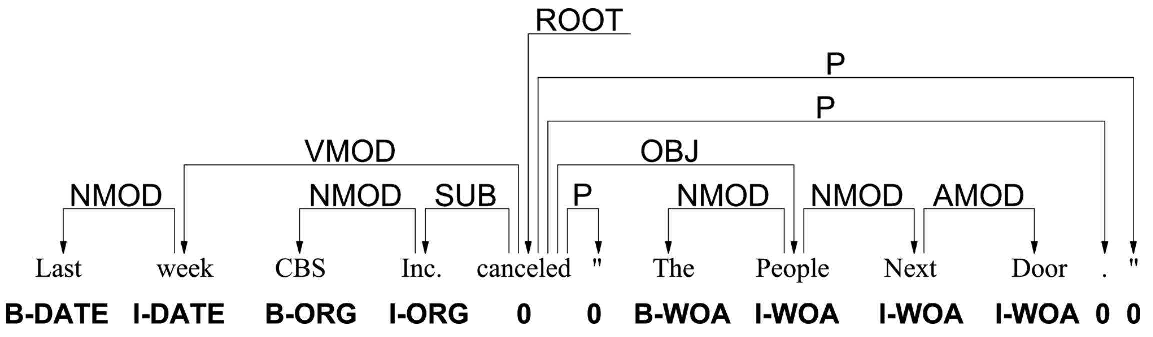


Figure 1. A dependency tree from the Penn Treebank, with additional entity annotation from the BBN corpus.

relation for the dependency graph, which consists of a set of labeled arcs (w_i, r, w_j), where $w_i, w_j \in W$ (the set of tokens), $d \in D$ (the set of dependencies). Given an input sentence s, the parser is initialized to $\langle \emptyset, s, \emptyset \rangle$, and terminates at configuration $\langle s, \emptyset, A \rangle$. There are three parsing schemata:

(1) Shift $\frac{\langle S,n|I,A\rangle}{\langle n|S,I,A\rangle}$

(2) $\mathrm{Right_r}$ $\frac{\langle s|S,n|I,A\rangle}{\langle S,n|I,A\cup\{(s,r,n)\}\rangle}$

(3) $\mathrm{Left_r}$ $\frac{\langle s|S,n|I,A\rangle}{\langle S,s|I,A\cup\{(n,r,s)\}\rangle}$

The Shift rule advances on the input; each $\mathrm{Left_r}$ and $\mathrm{Right_r}$ rule creates a link r between the next input token n and the top token on the stack s. For producing labeled dependencies the rules $\mathrm{Left_r}$ and $\mathrm{Right_r}$ are instantiated several times once for each dependency label.

Additional parsing actions (cf. (Attardi, 2006)) have been introduced for handling non-projective dependency trees: i.e., trees that cannot be drawn in the plane without crossing edges. However, they are not needed in the experiments reported here, because in the Penn Treebank used in our experiments dependencies are extracted without considering empty nodes and the resulting trees are all projective[2].

The pseudo code in Algorithm 1 reproduces schematically the parsing process.

The function $\mathrm{getContext}()$ extracts a vector of features $\mathbf{x}$ relative to the structure built up to that point from the context of the current token, i.e., from a subset of I, S and A. The step $\mathrm{estimateAction}()$ predicts a parsing action y, given a trained model α and $\mathbf{x}$. The final step $\mathrm{performAction}()$ updates the state according to the predicted parsing rule.

[2]Instead, the version of the Penn Treebank used for the CoNLL 2007 shared task includes also non-projective representations.

Algorithm 1: DeSR: Dependency Shift Reduce parser.

```
input: $s = w_1, w_2, ..., w_n$
begin
    $S \leftarrow \langle\rangle$
    $I \leftarrow \langle w_1, w_2, ..., w_n \rangle$
    $A \leftarrow \langle\rangle$
    while $I \neq \langle\rangle$ do
        $\mathbf{x} \leftarrow \mathrm{getContext}(S, I, A)$
        $y \leftarrow \mathrm{estimateAction}(\mathbf{x}, \alpha)$
        $\mathrm{performAction}(y, S, I, A)$
end
```

3.2 Features

The set of features used in this paper were chosen with a few simple experiments on the development data as a variant of a generic model. The only features of the tokens used are “Lemma”, “Pos” and “Dep”: “Lemma” refers to the morphologically simplified form of the token, “Pos” is the Part-of-Speech and “Dep” is the label on a dependency. “Child” refers to the child of a node (right or left): up to two furthest children of a node are considered. Table 1 lists which feature is extracted for which token: negative numbers refer to tokens on the stack, positive numbers refer to input tokens. As an example, POS(-1) is the Part-of-Speech of the token on the top of the stack, while Lemma(0) is the lemma of the next token in the input, PosLeftChild(-1) extracts the Part-of-Speech of the leftmost child of the token on the top of the stack, etc.

FEATURES	TOKEN					
	Stack		Input			
Lemma	-2	-1	0	1	2	3
Pos	-2	-1	0	1	2	3
LemmaLeftChild		-1	0			
PosLeftChild		-1	0			
DepLeftChild		-1	0			
LemmaRightChild		-1	0			
PosRightChild		-1	0			
DepRightChild		-1				
LemmaPrev			0			
PosSucc		-1				

Table 1. Configuration of the feature parameters used in the experiments.

3.3 Learning a parsing model with the perceptron

The problem of learning a parsing model can be framed as a classification task where each class $y_i \in \mathcal{Y}$ represents one of k possible parsing actions. Each of such actions is associated with a weight vector $\alpha_k \in \mathbb{R}^d$. Given a datapoint $\mathbf{x} \in \mathcal{X}$, a d-dimensional vector of binary features in the input space $\mathcal{X}$, a parsing action is chosen with a winner-take-all discriminant function:

$$\text{estimateAction}(\mathbf{x}, \alpha) = \arg\max_k f(\mathbf{x}, \alpha_k) \quad (4)$$

when using a linear classifier, such as the perceptron or SVM, $f(\mathbf{u}, \mathbf{v}) = \langle \mathbf{u}, \mathbf{v} \rangle$ is the inner product between vectors $\mathbf{u}$ and $\mathbf{v}$.

We learn the parameters α from the training data with the perceptron (Rosemblatt, 1958), in the online multiclass formulation of the algorithm (Crammer & Singer, 2003) with uniform negative updates. The perceptron has been used in previous work on dependency parsing by Carreras et al. (2006), with a parser based on Eisner's algorithm (Eisner, 2000), and also on incremental constituent parsing (Collins & Roark, 2006). Also the MST parser of McDonald uses a variant of the perceptron algorithm (McDonald, 2006). The choice is motivated by the simplicity and performance of perceptrons, which have proved competitive on a number of tasks; e.g., in shallow parsing, where perceptron's performance is comparable to that of Conditional Random Field models (Sha & Pereira, 2003).

The only adjustable parameter of the model is the number of instances T to use for training. We fixed T using the development portion of the data. In our experiments, the best value is between 20 and 30 times the size of the training data. To regularize the model we take as the final model the average of all weight vectors posited during training (Collins, 2002). Algorithm 2 illustrates the perceptron learning procedure. The final average model can be computed efficiently during training without storing the individual α vectors (e.g., see (Ciaramita & Johnson, 2003)).

Algorithm 2: Average multiclass perceptron

```
input : $\mathcal{S} = (\mathbf{x}_i, y_i)^N; \alpha_k^0 = \vec{0}, \forall k \in \mathcal{Y}$
for $t = 1$ to $T$ do
    choose $j$
    $E^t = \{r \in \mathcal{Y} : \langle \mathbf{x}_j, \alpha_r^t \rangle \geq \langle \mathbf{x}_j, \alpha_{y_j}^t \rangle\}$
    if $|E^t| > 0$ then
        $\alpha_r^{t+1} = \alpha_r^t - \frac{\mathbf{x}_j}{|E^t|}, \forall r \in E^t$
        $\alpha_{y_j}^{t+1} = \alpha_{y_j}^t + \mathbf{x}_j$
output: $\alpha_k = \frac{1}{T} \sum_t \alpha_k^t, \forall k \in \mathcal{Y}$
```

3.4 Higher-order feature spaces

Yamada and Matsumoto (2003) and McDonald and Pereira (2006) have shown that higher-order feature representations and modeling can improve parsing accuracy, although at significant computational costs. To make SVM training feasible in the dual model with polynomial kernels, Yamada and Matsumoto split the training data into several sets, based on POS tags, and train a parsing model for each set. McDonald and Pereira's second-order MST parser has $O(n^3)$ complexity, while for handling non-projective trees, otherwise an NP-hard problem, the parser resorts to an approximate algorithm. Here we discuss how the feature representation can be enriched to improve parsing while maintaining the simplicity of the shift-reduce architecture, and performing discriminative learning without partitioning the training data.

The linear classifier (see Equation 4) learned with the perceptron is inherently limited in the types of solutions it can learn. As originally pointed out by Minsky and Papert (1969), there are problems which require non-linear solutions that cannot be learned by such models. A simple workaround this limitation relies on feature maps $\mathbf{\Phi} : \mathbb{R}^d \rightarrow \mathbb{R}^h$ that map the input vectors $\mathbf{x} \in \mathcal{X}$ into some higher h-dimensional representation $\mathbf{\Phi}(\mathcal{X}) \subset \mathrm{I\!R}^h$, the feature space. The feature space can represent, for example, all combinations of individual features in the input space. We define a feature map which extracts all second order features of the form $x_i x_j$; i.e., $\mathbf{\Phi}(\mathbf{x}) = (x_i, x_j | i = 1, ..., d, j = i, ..., d)$. The linear perceptron working in $\mathbf{\Phi}(\mathcal{X})$ effectively implements a non-linear classifier in the original input space $\mathcal{X}$. One shortcoming of this approach is that it inflates considerably the feature representation and might not scale. In general, the number of features of degree g over an input space of dimension d is $\binom{d+g-1}{g}$. In practice, a second-order feature map can be handled with reasonable efficiency by the perceptron. We call this the 2nd-order model, which uses a modified scoring function:

$$g(\mathbf{x}, \alpha_k) = f(\Phi(\mathbf{x}), \alpha_k) \tag{5}$$

where also α_k is h-dimensional. The proposed feature map is equivalent to a polynomial kernel function of degree two. Yamada and Matsumoto (2003) have shown that the degree two polynomial kernel has superior accuracy than the linear model and polynomial kernels of higher degrees. However, using the dual model is not always practical for dependency parsing. The discriminant function of the dual model is defined as:

$$f'(\mathbf{x}, \alpha) = \arg\max_k \sum_{i=1}^{N} \alpha_{k,i} \langle \mathbf{x}, \mathbf{x}_i \rangle^g \tag{6}$$

where the weights α are associated with class-instance pairs rather than class-feature pairs. With respect to the discriminant function of equation (4) there is an additional summation. In principle, the inner products can be cached in a Kernel matrix to speed up training.

There are two shortcomings to using such a model in dependency parsing. First, if the amount of training data is large it might not be feasible to store the Kernel matrix; which for a dataset of size N requires $O(N^3)$ computations and $O(N^2)$ space. As an example, the number of training instances N in the Penn Treebank is over 1.8 million, caching the Kernel matrix would require several Terabytes of space. The second shortcoming is independent of training. In predicting a tree for unseen sentences the model will have to recompute the inner products between the observation and all the support vectors; i.e., all class-instance pairs with $\alpha_{k,i} > 0$. The second-order feature map with the perceptron is more efficient and allows faster training and prediction. Training a single parsing model avoids a potential loss of accuracy that occurs when using the technique of partitioning the training data according to the POS. Inaccurate predictions of the POS can affect significantly the accuracy of the actions predicted, while the single model is more robust, since the POS is just one of the many features used in prediction.

4 Semantic features

Semantic information is used implicitly in parsing. For example, conditioning on lexical heads provides a source of semantic information. There have been a few attempts at using semantic information more explicitly. Charniak's 1997 parser (1997), defined probability estimates backed off to word clusters. Collins and Koo (Collins & Koo, 2005) introduced an improved reranking model for parsing which includes a hidden layer of semantic features. Yi and Palmer (2005) retrained a constituent parser in which phrases were annotated with argument information to improve SRL, however this didn't improve over the output of the basic parser.

In recent years there has been a significant amount of work on semantic annotation tasks such as named-entity recognition, semantic role labeling and relation extraction. There is evidence that dependency and constituent parsing can be helpful in these and other tasks; e.g., by means of tree kernels in question classification and semantic role labeling (Zhang & Lee, 2003; Moschitti, 2006).

It is natural to ask if also the opposite holds: whether semantic annotations can be used to improve parsing. In particular, it would be interesting to know if entity-like tags can be used for this purpose. One reason for this is that entity tagging is efficient and does not seem to need parsing for achieving top performance. Beyond improving traditional parsing, independently learned semantic tags might be helpful in adapting a parser to a new domain. To the best of our knowledge, no evidence has been produced yet that annotated semantic information can improve parsing. In the following we investigate adding entity tags as features of our parser.

4.1 BBN Entity corpus

The BBN corpus (BBN, 2005) supplements the Wall Street Journal Penn Treebank with annotation of a large set of entity types. The corpus includes annotation of 12 named entity types (Person, Facility, Organization, GPE, Location, Nationality, Product, Event, Work of Art, Law, Language, and Contact-Info), nine nominal entity types (Person, Facility, Organization, GPE, Product, Plant, Animal, Substance, Disease and Game), and seven numeric types (Date, Time, Percent, Money, Quantity, Ordinal and Cardinal). Several of these types are further divided into subtypes[3]. This corpus provides adequate support for experimenting semantic features for parsing.

Figure 1 illustrates the annotation layer provided by the BBN corpus[4]. It is interesting to notice one apparent property of the combination of semantic tags and dependencies. When we consider segments composed of several words there is exactly one dependency connecting a token outside the segment with a token inside the segment; e.g., "CBS Inc." is connected outside only through the token "Inc.", the subject of the main verb. With respect to the rest of the tree, segments tend to form units, with their own internal structure. Intuitively, this information seems relevant for parsing. This locally-structured patterns could help particularly simple algorithms like ours, which have limited knowledge of the global structure being built.

Table 2 lists the 40 most frequent categories in sections 2 to 21 of the BBN corpus, and the percentage of all entities they represent – together more than 97%. Sections 2-21 are comprised of 949,853 tokens, 23.5% of the tokens have a non-null BBN entity tag, on average there is one tagged token every four. The total number of entities is 139,029, 70.5% of which are named entities and nominal concepts, 17% are numerical types and the remaining 12.5% describe time entities.

We designed three new features which extract simple properties of entities from the semantic annotation information:

[3]BBN Corpus documentation.

[4]The full label for "ORG" is "ORG:Corporation", and "WOA" stands for "WorkOfArt:Other".

	TOKEN				
FEATURES	Stack		Input		
AS-0 = EOS+BIO+TAG			0		
AS-1 = EOS+BIO+TAG		-1	0	1	
AS-2 = EOS+BIO+TAG	-2	-1	0	1	2
EOS	-2	-1	0	1	2
BIO	-2	-1	0	1	2
TAG	-2	-1	0	1	2

Table 3. Additional configurations for the models with BBN entity features.

- EOS: Distance to the end of the segment; e.g., EOS("Last") = 1, EOS("canceled") = 0;
- BIO: The first character of the BBN label for a token; e.g., BIO("CBS") = "B", and BIO("canceled") = 0;
- TAG: Full BBN tag for the token; e.g., TAG("CBS") = "B-ORG:Corporation", TAG("week") = "I-DATE".

The feature EOS provides information about the relative position of the token within a segment with respect to the end of the segment. The feature BIO discriminates tokens with no semantic annotation associated, from tokens within a segment and token which start a segment. Finally the feature TAG identifies the full semantic tag associated with the token. With respect to the former two features this bears the most fine-grained semantics. Table 3 summarizes six additional models we implemented. The first three use all additional features together, applied to different sets of tokens, while the last three apply only one feature, on top of the base model, relative to the next token in the input, the following two tokens in the input, and the previous two tokens on the stack.

4.2 Corpus pre-processing

The original BBN corpus has its own tokenization which often does not reflect the Penn Treebank tokenization; e.g., when an entity intersects an hyphenated compound, thus "third-highest" becomes "third$_{ORDINAL}$ - highest". This is problematic for combining entity annotation and dependency trees. Since our main focus is parsing we re-aligned the BBN Corpus with the Treebank tokenization. Thus, for example, when an entity splits a Treebank token we extend the entity boundary to contain

WSJ-BBN Corpus Categories							
Tag	%	Tag	%	Tag	%	Tag	%
PER_DESC	15.5	ORG:CORP	13.7	DATE:DATE	9.2	ORG_DESC:CORP	8.9
PERSON	8.13	MONEY	6.5	CARDINAL	6.0	PERCENT	3.5
GPE:CITY	3.12	GPE:COUNTRY	2.9	ORG:GOV	2.6	NORP:NATION-TY	1.9
DATE:DURATION	1.8	GPE:PROVINCE	1.5	ORG_DESC:GOV	1.4	FAC_DESC:BLDG	1.1
ORG:OTHER	0.7	PROD_DESC:VEHICLE	0.7	ORG_DESC:OTHER	0.6	ORDINAL	0.6
TIME	0.5	GPE_DESC:COUNTRY	0.5	SUBST:OTHER	0.5	SUBST:FOOD	0.5
DATE:OTHER	0.4	NORP:POLITICAL	0.4	DATE:AGE	0.4	LOC:REGION	0.3
SUBST:CHEM	0.3	WOA:OTHER	0.3	FAC_DESC:OTHER	0.3	SUBST:DRUG	0.3
ANIMAL	0.3	GPE_DESC:PROVINCE	0.2	PROD:VEHICLE	0.2	GPE_DESC:CITY	0.2
PRODUCT:OTHER	0.2	LAW	0.2	ORG:POLITICAL	0.2	ORG:EDU	0.2

Table 2. The 40 most frequent labels in sections 2 to 21 of the Wall Street Journal BBN Corpus and the percentage of tags occurrences.

the whole original Treebank token, thus obtaining "third-highest$_{ORDINAL}$" in the example above.

4.3 Semantic tagger

We treated semantic tags as POS tags. A tagger was trained on the BBN gold standard annotation and used it to annotate development and evaluation data. We briefly describe the tagger (see (Ciaramita & Altun, 2006) for more details), a Hidden Markov Model trained with the perceptron algorithm introduced in (Collins, 2002). The tagger uses Viterbi decoding. Label to label dependencies are limited to the previous tag (first order HMM). A generic feature set for NER based on words, lemmas, POS tags, and word shape features was used.

The tagger is trained on sections 2-21 of the BBN corpus. As before, section 22 of the BBN corpus is used for choosing the perceptron's parameter T. The tagger's model is regularized as described for Algorithm 2. The full BBN tagset is comprised of 105 classes organized hierarchically, we ignored the hierarchical organization and treated each tag as an independent class in the standard BIO encoding. The tagger evaluated on section 23 achieves an F-score of 86.8%. The part of speech for the evaluation/development sections was produced with TreeTagger. As a final remark we notice that the tagger's complexity, linear in the length of the sentence, preserves the parser's complexity.

5 Parsing experiments

5.1 Data and setup

We used the standard partitions of the Wall Street Journal Penn Treebank (Marcus et al., 1993); i.e., sections 2-21 for training, section 22 for development and section 23 for evaluation. The constituent trees were transformed into dependency trees by means of a program created by Joakim Nivre that implements the rules proposed by Yamada and Matsumoto, which in turn are based on the head rules of Collins' parser (Collins, 1999)[5]. The lemma for each token was produced using the "morph" function of the WordNet (Fellbaum, 1998) library[6]. The data in the WSJ sections 22 and 23, both for the parser and for the semantic tagger, was POS-tagged using TreeTagger[7], which has an accuracy of 97.0% on section 23.

Training a parsing model on the Wall Street Journal requires a set of 22 classes: 10 of the 11 labels in the dependency corpus generated from the Penn Treebank (e.g., subj, obj, sbar, vmod, nmod, root, etc.) are paired with both a Left and Right actions. In addition, there is in one rule for the "root" label and one for the Shift action. The total number of features found in training ranges from two hundred thousand for the 1st-order model to approximately 20 million of the 2nd-order models.

We evaluated several models, each trained with 1st-order and 2nd-order features. The base model (BASE) only uses the traditional set of features (cf. Table 1). Models EOS, BIO and TAG each use only one type of semantic feature with the configuration described in Table 3. Models AS-0, AS-1, and AS-2 use all three semantic features for the token on the stack in AS-0, plus the previous token on the stack and the new token in the input in AS-1, plus an addi-

[5]The script is available from http://w3.msi.vxu.se/%7enivre/research/Penn2Malt.html

[6]http://wordnet.princeton.edu

[7]TreeTagger is available from http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/

	1st-order scores				2nd-order scores			
DeSR MODEL	LAS	UAS	Imp	LAC	LAS	UAS	Imp	LAC
BASE	84.01	85.56	-	88.24	89.20	90.55	-	92.22
EOS	84.89	86.37	+5.6	88.94	89.36	90.64	+1.0	92.37
BIO	84.95	86.37	+6.6	89.06	89.63	90.89	+3.6	92.55
TAG	84.76	86.26	+4.8	88.80	89.54	90.81	+2.8	92.55
AS-0	84.40	85.95	+2.7	88.38	89.41	90.72	+1.8	92.38
AS-1	85.13	86.52	+6.6	89.11	89.57	90.77	+2.3	92.49
AS-2	85.32	86.71	+8.0	89.25	**89.87**	**91.10**	+5.8	**92.68**

Table 4. Results of the different models on WSJ section 23 using the CoNLL scores Labeled attachment score (LAS), Unlabeled attachment score (UAS), and Label accuracy score (LAC). The column labeled "Imp" reports the improvement in terms of relative error reduction with respect to the BASE model for the UAS score. In bold the best results.

tional token from the stack and an additional token from the input for AS-2 (cf. Table 3).

5.2 Results of 2nd-order models

Table 4 summarizes the results of all experiments. We report the following scores, obtained with the CoNLL-X scoring script: labeled attachment score (LAS), unlabeled attachment score (UAS) and label accuracy score (LAC). For the UAS score, the most frequently reported, we include the improvement in relative error reduction.

The 2nd-order base model improves on all measures over the 1st-order model by approximately 5%. The UAS score is 90.55%, with an improvement of 4.9%. The magnitude of the improvement is remarkable and reflects the 4.6% improvement that Yamada and Matsumoto (Yamada & Matsumoto, 2003) report going from the linear SVM to the polynomial of degree two. Our base model's accuracy (90.55% UAS) compares well with the accuracy of the parsers based on the polynomial kernel trained with SVM of Yamada and Matsumoto (UAS 90.3%), and Hall et al. (2006) (UAS 89.4%). We notice in particular that, given the lack of non-projective cases/rules, the parser of Hall et al. (2006) is almost identical to our parser, hence the difference in accuracy (+1.1%) might effectively be due to a better classifier. Yamada & Matsumoto's parser is slightly more complex than our parser, and has quadratic worst-case complexity. Overall, the accuracy of the 2nd-order parser is comparable to that of the 1st-order MST parser (90.7%).

There is no direct evidence that our perceptron produces better classifiers than SVM. Rather, the pattern of results produced by the perceptron seems comparable to that of SVM (Yamada & Matsumoto, 2003). This is a useful finding in itself, given that the former is more efficient: perceptron's update is linear while SVM solves a quadratic problem at each update. However, one major difference between the two approaches lies in the fact that learning with the primal model does not require splitting the model by Part-of-Speech, or other means. As a consequence, beyond the greater simplicity, our method might benefit from not depending so strongly on the quality of POS tagging. POS information is encoded as a feature and contributes its weight to the selection of the parsing action, together with all additionally available information. In the SVM-trained methods the model that makes the prediction for the parsing rule is essentially chosen by an oracle, the prediction of the POS tagger. Furthermore, it might be argued that learning a single model makes a better use of the training data by exploiting the correlations between all datapoints, while in the dual split-training case the interaction is limited to datapoints in the same partition. In any case, second-order feature maps could be used also with SVM or other classifiers. The advantage of using the perceptron lies in the unchallenged accuracy/efficiency trade-off. Finally, we recall that training in the primal model can be performed fully on-line without affecting the resulting model nor the complexity of the algorithm.

5.3 Results of models with semantic features

All models based on semantic features improve over the base model on all measures. The best configura-

Parser	UAS
Hall et al. '06	89.4
Yamada & Matsumoto '03	90.3
DeSR	90.55
McDonald & Pereira 1st-order MST	90.7
DeSR AS-2	91.1
McDonald & Pereira 2nd-order MST	91.5
Sagae & Lavie '06	92.7

Table 5. Comparison of main results on the Penn Treebank dataset.

tion is that of model AS-2 which extracts all semantic features from the widest context. In the 1st-order AS-2 model the improvement, 86.71% UAS (+8% relative error reduction) is more marked than in the 2nd-order AS-2 model, 91.1% UAS (+5.8% error reduction). A possible simple exaplanation is that some information captured by the semantic features is correlated with other higher-order features which do not occur in the 1st-order encoding. Overall the accuracy of the DeSR parser with semantic information is slightly inferior to that of the second-order MST parser (McDonald & Pereira, 2006) (91.5% UAS). The best result on this dataset to date (92.7% UAS) is that of Sagae and Lavie (Sagae & Lavie, 2006) who use a parser which combines the predictions of several pre-existing parsers, including McDonald's and Nivre's parsers. Table 5 lists the main results to date on the version of the Penn Treebank for dependency parsing task used in this paper.

In Table 4 we also evaluate the gain obtained by adding one semantic feature type at a time (cf. rows EOS/BIO/TAG). These results show that all semantic features provide some improvement (with the dubious case of EOS in the 2nd-order model). The BIO encoding seems to produce the most accurate features. This could be promising because it suggests that the benefit does not depend only on the specific tags, but that the segmentation in itself is important. Hence tagging could improve the adaptation of parsers to new domains even if only generic tagging methods are available.

5.4 Remarks on efficiency

All experiments were performed on a 2.4GHz AMD Opteron CPU machine with 32GB RAM. The 2nd-order parser uses almost 3GB of memory. While

	Parsing time/sec	
Parser	English	Chinese
MST 2n-order	97.52	59.05
MST 1st-order	76.62	49.13
DeSR	36.90	21.22

Table 6. Parsing times for the CoNNL 2007 English and Chinese datasets for MST and DeSR.

it is several times slower and larger than the 1st-order model[8] the 2nd-order model performance is still competitive. It takes 3 minutes (user time) to parse section 23, POS tagging included. In training, the model takes about 1 hour to process the full dataset once. As a comparison, Hall et al. (2006) reports 1.5 hours for training the partitioned SVM model and 10 minutes for parsing the evaluation set on the same Penn Treebank data. We also compared directly the parsing time of our parser with that of the MST parser using the version 0.4.3 of MST-Parser[9]. For these experiments we used two datasets from the CoNLL 2007 shared task for English and Chinese. Table 6 reports the times, in seconds, to parse the test sets for these languages on a 3.3GHz Xeon machine with 4 GB Ram, of the MST 1st and 2nd-order parser and DeSR parser (without semantic features).

The architecture of the model presented here offers several options for optimization. For example, implementing the α models with full vectors rather than hash tables speeds up parsing by a factor of three, at the expense of memory. Alternatively, memory load in training can be reduced, at the expense of time, by using on-line training. However, the most valuable option for space need reduction might be to filter out low-frequency second-order features. Since the frequency of such features seems to follow a power law distribution, this reduces significantly the feature space size even for low thresholds at small accuracy expense. In this paper however we focused on the full model, no approximations were required to run the experiments.

[8]The 1st-order parser takes 7 seconds (user time) to process Section 23.

[9]Available from sourceforge.net.

6 Conclusion

We explored the design space of a dependency parser by modeling and extending the feature representation, while adopting one of the simplest parsing architecture: a single-pass deterministic shift-reduce algorithm trained with a regularized multiclass perceptron. We showed that with the perceptron it is possible to adopt higher-order feature maps equivalent to polynomial kernels without need of approximating the model (although this remains an option for optimization). The resulting models achieve accuracies comparable (or better) to more complex architectures based on dual SVM training, and faster parsing on unseen data. With respect to learning, it is possible that more sophisticated formulations of the perceptron (e.g. MIRA (Crammer & Singer, 2003)) could provide further gains in accuracy, as shown with the MST parser (McDonald et al., 2005).

We also experimented with novel types of semantic features, extracted from the annotations produced by an entity tagger trained on the BBN corpus. This model further improves over the standard model yielding an additional 5.8% relative error reduction. Although the magnitude of the improvement is not striking, to the best of our knowledge this is the first encouraging evidence that annotated semantic information can improve parsing and suggests several options for further research. For example, this finding might indicate that this type of approach, which combines semantic tagging and parsing, is viable for the adaptation of parsing to new domains for which semantic taggers exist. Semantic features could be also easily included in other types of dependency parsing algorithms, e.g., MST, and in current methods for constituent parse reranking (Collins, 2000; Charniak & Johnson, 2005).

For future research several issues concerning the semantic features could be tackled. We notice that more complex semantic features can be designed and evaluated. For example, it might be useful to guess the "head" of segments with simple heuristics, i.e., the guess the node which is more likely to connect the segment with the rest of the tree, which all internal components of the entity depend upon. It would be also interesting to extract semantic features from taggers trained on different datasets and based on different tagsets.

Acknowledgments

The first author would like to thank Thomas Hofmann for useful inputs concerning the presentation of the issue of higher-order feature representations of Section 3.4. We would also like to thank Brian Roark and the anonymous reviewers for useful comments and pointers to related work.

References

G. Attardi. 2006. Experiments with a Multilanguage Non-Projective Dependency Parser. In *Proceedings of CoNNL-X 2006*.

BBN. 2005. BBN Pronoun Coreference and Entity Type Corpus. Linguistic Data Consortium (LDC) catalog number LDC2005T33.

S. Buchholz and E. Marsi. 2006. Introduction to CoNNL-X Shared Task on Multilingual Dependency Parsing. In *Proceedings of CoNNL-X 2006*.

X. Carreras and L. Màrquez. 2005. Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling. In *Proceedings of CoNLL 2005*.

X. Carreras, M. Surdeanu, and L. Màrquez. 2006 Projective Dependency Parsing with Perceptron. In *Proceedings of CoNLL-X*.

E. Charniak. 1997. Statistical Parsing with a Context-Free Grammar and Word Statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence AAAI*.

E. Charniak and M. Johnson. 2005. Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking. In *Proceedings of ACL 2005*.

M. Ciaramita and Y. Altun. 2006. Broad-Coverage Sense Disambiguation and Information Extraction with a Supersense Sequence Tagger. In *Proceedings of EMNLP 2006*.

M. Ciaramita and M. Johnson. 2003. Supersense Tagging of Unknown Nouns in WordNet. In *Proceedings of EMNLP 2003*.

M. Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. Thesis, University of Pennsylvania.

M. Collins. 2000. Discriminative Reranking for Natural Language Parsing. In *Proceedings of ICML 2000*.

M. Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of EMNLP 2002*.

M. Collins and T. Koo. 2005. Hidden-Variable Models for Discriminative Reranking. In *P*roceedings of EMNLP 2005.

M. Collins and B. Roark. 2004. Incremental Parsing with the Perceptron Algorithm. In *Proceedings of ACL 2004*.

K. Crammer and Y. Singer. 2003. *Ultraconservative Online Algorithms for Multiclass Problems.* Journal of Machine Learning Research 3: pp.951-991.

A. Culotta and J. Sorensen. 2004. Dependency Tree Kernels for Relation Extraction. In *Proceedings of ACL 2004*.

Y. Ding and M. Palmer. 2005. Machine Translation using Probabilistic Synchronous Dependency Insertion Grammars. In *Proceedings of ACL 2005*.

J. Eisner. 2000. Bilexical Grammars and their Cubic-Time Parsing Algorithms. In H.C. Bunt and A. Nijholt, eds. *New Developments in Natural Language Parsing*, pp. 29-62. Kluwer Academic Publishers.

C. Fellbaum. 1998. *WordNet: An Electronic Lexical Database* MIT Press, Cambridge, MA. 1969.

J. Hall, J. Nivre and J. Nilsson. 2006. Discriminative Classifiers for Deterministic Dependency Parsing. In *P*roceedings of the COLING/ACL 2006.

T. Kalt. 2004. Induction of Greedy Controllers for Deterministic Treebank Parsers. In *Proceedings of EMNLP 2004*.

M. Marcus, B. Santorini and M. Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2): pp. 313-330.

R. McDonald. 2006. *Discriminative Training and Spanning Tree Algorithms for Dependency Parsing*. Ph.D. Thesis, University of Pennsylvania.

R. McDonald, F. Pereira, K. Ribarov and J. Hajič. 2005. Non-projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of HLT-EMNLP 2005*.

R. McDonald and F. Pereira. 2006. Online Learning of Approximate Dependency Parsing Algorithms. In *Proceedings of EACL 2006*.

M.L. Minsky and S.A. Papert. 1969. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, MA. 1969.

A. Moschitti. 2006. Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees. In *Proceedings of ECML 2006*.

J. Nivre. 2004. Incrementality in Deterministic Dependency Parsing. In *Incremental Parsing: Bringing Engineering and Cognition Together. Workshop at ACL-2004*.Spain, 50-57.

J. Nivre, J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel and D. Yuret. 2007. The CoNLL 2007 Shared Task on Dependency Parsing, In Proceedings of EMNLP-CoNLL 2007.

J. Nivre and M. Scholz. 2004. Deterministic Dependency Parsing of English Text. In *Proceedings of COLING 2004*.

V. Punyakanok, D. Roth, and W. Yih. 2005. The Necessity of Syntactic Parsing for Semantic Role Labeling. In *Proceedings of IJCAI 2005*.

F. Rosemblatt. 1958. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psych. Rev.*, 68: pp. 386-407.

K. Sagae and A. Lavie. 2005. Parser Combination by Reparsing. In *Proceedings of HLT-NAACL 2006*.

F. Sha and F. Pereira. 2003. Shallow Parsing with Conditional Random Fields. In *Proceedings of HLT-NAACL 2003*.

H. Yamada and Y. Matsumoto. 2003. Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of the Eighth International Workshop on Parsing Technologies. Nancy, France*.

S. Yi and M. Palmer. 2005. The Integration of Syntactic Parsing and Semantic Role Labeling. In *Proceedings of CoNLL 2005*.

A. Wong and D. Wu. 1999. Learning a Lightweight Deterministic Parser. In *Proceedings of EUROSPEECH 1999*.

D. Zhang and W.S. Less. 2003. Question Classification using Support Vector Machines. In *Proceedings of SIGIR 2003*.

A Latent Variable Model for Generative Dependency Parsing

Ivan Titov
University of Geneva
24, rue Général Dufour
CH-1211 Genève 4, Switzerland
ivan.titov@cui.unige.ch

James Henderson
University of Edinburgh
2 Buccleuch Place
Edinburgh EH8 9LW, United Kingdom
james.henderson@ed.ac.uk

Abstract

We propose a generative dependency parsing model which uses binary latent variables to induce conditioning features. To define this model we use a recently proposed class of Bayesian Networks for structured prediction, Incremental Sigmoid Belief Networks. We demonstrate that the proposed model achieves state-of-the-art results on three different languages. We also demonstrate that the features induced by the ISBN's latent variables are crucial to this success, and show that the proposed model is particularly good on long dependencies.

1 Introduction

Dependency parsing has been a topic of active research in natural language processing during the last several years. The CoNLL-X shared task (Buchholz and Marsi, 2006) made a wide selection of standardized treebanks for different languages available for the research community and allowed for easy comparison between various statistical methods on a standardized benchmark. One of the surprising things discovered by this evaluation is that the best results are achieved by methods which are quite different from state-of-the-art models for constituent parsing, e.g. the deterministic parsing method of (Nivre et al., 2006) and the minimum spanning tree parser of (McDonald et al., 2006). All the most accurate dependency parsing models are fully discriminative, unlike constituent parsing where all the state of the art methods have a generative component (Charniak and Johnson, 2005; Henderson, 2004; Collins, 2000). Another surprising thing is the lack of latent variable models among the methods used in the shared task. Latent variable models would allow complex features to be induced automatically, which would be highly desirable in multilingual parsing, where manual feature selection might be very difficult and time consuming, especially for languages unknown to the parser developer.

In this paper we propose a generative latent variable model for dependency parsing. It is based on Incremental Sigmoid Belief Networks (ISBNs), a class of directed graphical model for structure prediction problems recently proposed in (Titov and Henderson, 2007), where they were demonstrated to achieve competitive results on the constituent parsing task. As discussed in (Titov and Henderson, 2007), computing the conditional probabilities which we need for parsing is in general intractable with ISBNs, but they can be approximated efficiently in several ways. In particular, the neural network constituent parsers in (Henderson, 2003) and (Henderson, 2004) can be regarded as coarse approximations to their corresponding ISBN model.

ISBNs use history-based probability models. The most common approach to handling the unbounded nature of the parse histories in these models is to choose a pre-defined set of features which can be unambiguously derived from the history (e.g. (Charniak, 2000; Collins, 1999; Nivre et al., 2004)). Decision probabilities are then assumed to be independent of all information not represented by this finite set of features. ISBNs instead use a vector of binary latent variables to encode the information about the parser history. This history vector is similar to the hidden state of a Hidden Markov Model. But unlike the graphical model for an HMM, which specifies conditional dependency edges only between adjacent states in the sequence, the ISBN graphical model can specify conditional dependency edges between states which are arbitrarily far apart in the parse history. The source state of such an edge is determined by the partial output structure built at the time of the destination state, thereby allowing the conditional dependency edges to be appropriate for the structural nature of the problem being modeled. This structure sensitivity is possible because ISBNs are a constrained form of switching model (Murphy, 2002), where each output decision switches the model structure used for the remaining decisions.

We build an ISBN model of dependency parsing using the parsing order proposed in (Nivre et al., 2004). However, instead of performing deterministic parsing as in (Nivre et al., 2004), we use this ordering to define a generative history-based model, by integrating word prediction operations into the set of parser actions. Then we propose a simple, language independent set of relations which determine how latent variable vectors are interconnected by conditional dependency edges in the ISBN model. ISBNs also condition the latent variable vectors on a set of explicit features, which we vary in the experiments.

In experiments we evaluate both the performance of the ISBN dependency parser compared to previous work, and the ability of the ISBN model to induce complex history features. Our model achieves state-of-the-art performance on the languages we test, significantly outperforming the model of (Nivre et al., 2006) on two languages out of three and demonstrating about the same results on the third. In order to test the model's feature induction abilities, we train models with two different sets of explicit conditioning features: the feature set individually tuned by (Nivre et al., 2006) for each considered language, and a minimal set of local features. These models achieve comparable accuracy, unlike with the discriminative SVM-based approach of (Nivre et al., 2006), where careful feature selection appears to be crucial. We also conduct a controlled experiment where we used the tuned features of (Nivre et al., 2006) but disable the feature induction abilities of our model by elimination of the edges connecting latent state vectors. This restricted model achieves far worse results, showing that it is exactly the capacity of ISBNs to induce history features which is the key to its success. It also motivates further research into how feature induction techniques can be exploited in discriminative parsing methods.

We analyze how the relation accuracy changes with the length of the head-dependent relation, demonstrating that our model very significantly outperforms the state-of-the-art baseline of (Nivre et al., 2006) on long dependencies. Additional experiments suggest that both feature induction abilities and use of the beam search contribute to this improvement.

The fact that our model defines a probability model over parse trees, unlike the previous state-of-the-art methods (Nivre et al., 2006; McDonald et al., 2006), makes it easier to use this model in applications which require probability estimates, e.g. in language processing pipelines. Also, as with any generative model, it may be easy to improve the parser's accuracy by using discriminative retraining techniques (Henderson, 2004) or data-defined kernels (Henderson and Titov, 2005), with or even without introduction of any additional linguistic features. In addition, there are some applications, such as language modeling, which require generative models. Another advantage of generative models is that they do not suffer from the label bias problems (Bottou, 1991), which is a potential problem for conditional or deterministic history-based models, such as (Nivre et al., 2004).

In the remainder of this paper, we will first review general ISBNs and how they can be approximated. Then we will define the generative parsing model, based on the algorithm of (Nivre et al., 2004), and propose an ISBN for this model. The empirical part of the paper then evaluates both the overall accuracy of this method and the importance of the model's capacity to induce features. Additional related work will be discussed in the last section before concluding.

2 The Latent Variable Architecture

In this section we will begin by briefly introducing the class of graphical models we will be using, Incremental Sigmoid Belief Networks (Titov and Henderson, 2007). ISBNs are designed specifically for modeling structured data. They are latent variable models which are not tractable to compute exactly, but two approximations exist which have been shown to be effective for constituent parsing (Titov and Henderson, 2007). Finally, we present how these approximations can be trained.

2.1 Incremental Sigmoid Belief Networks

An ISBN is a form of Sigmoid Belief Network (SBN) (Neal, 1992). SBNs are Bayesian Networks with binary variables and conditional probability distributions in the form:

$$P(S_i = 1|Par(S_i)) = \sigma(\sum_{S_j \in Par(S_i)} J_{ij} S_j),$$

where S_i are the variables, $Par(S_i)$ are the variables which S_i depends on (its parents), σ denotes the logistic sigmoid function, and J_{ij} is the weight for the edge from variable S_j to variable S_i in the graphical model. SBNs are similar to feed-forward neural networks, but unlike neural networks, SBNs have a precise probabilistic semantics for their hidden variables. ISBNs are based on a generalized version of SBNs where variables with any range of discrete values are allowed. The normalized exponential function ('soft-max') is used to define the conditional probability distributions at these nodes.

To extend SBNs for processing arbitrarily long sequences, such as a parser's sequence of decisions $D^1, ..., D^m$, SBNs are extended to a form of Dynamic Bayesian Network (DBN). In DBNs, a new set of variables is instantiated for each position in the sequence, but the edges and weights are the same for each position in the sequence. The edges which connect variables instantiated for different positions must be directed forward in the sequence, thereby allowing a temporal interpretation of the sequence.

Incremental Sigmoid Belief Networks (Titov and Henderson, 2007) differ from simple dynamic SBNs in that they allow the model structure to depend on the output variable values. Specifically, a decision is allowed to effect the placement of any edge whose destination is after the decision. This results in a form of switching model (Murphy, 2002), where each decision switches the model structure used for the remaining decisions. The incoming edges for a given position are a discrete function of the sequence of decisions which precede that position. This makes the ISBN an "incremental" model, not just a dynamic model. The structure of the model is determined incrementally as the decision sequence proceeds.

ISBNs are designed to allow the model structure to depend on the output values without overly complicating the inference of the desired conditional probabilities $P(D^t|D^1, \ldots, D^{t-1})$, the probability of the next decision given the history of previous decisions. In particular, it is never necessary to sum over all possible model structures, which in general would make inference intractable.

2.2 Modeling Structures with ISBNs

ISBNs are designed for modeling structured data where the output structure is not given as part of the input. In dependency parsing, this means they can model the probability of an output dependency structure when the input only specifies the sequence of words (i.e. parsing). The difficulty with such problems is that the statistical dependencies in the dependency structure are local in the structure, and not necessarily local in the word sequence. ISBNs allow us to capture these statistical dependencies in the model structure by having model edges depend on the output variables which specify the dependency structure. For example, if an output specifies that there is a dependency arc from word w_i to word w_j, then any future decision involving w_j can directly depend on its head w_i. This allows the head w_i to be treated as local to the dependent w_j even if they are far apart in the sentence.

This structurally-defined notion of locality is particularly important for the model's latent variables. When the structurally-defined model edges connect latent variables, information can be propagated between latent variables, thereby providing an even larger structural domain of locality than that provided by single edges. This provides a potentially powerful form of feature induction, which is nonetheless biased toward a notion of locality which is appropriate for the structure of the problem.

2.3 Approximating ISBNs

(Titov and Henderson, 2007) proposes two approximations for inference in ISBNs, both based on variational methods. The main idea of variational methods (Jordan et al., 1999) is, roughly, to construct a tractable approximate model with a number of free parameters. The values of the free parameters are set so that the resulting approximate model is as close as possible to the original graphical model for a given inference problem.

The simplest example of a variation method is the mean field method, which uses a fully factorized distribution $Q(H|V) = \prod_i Q_i(h_i|V)$ as the approximate model, where V are the visible (i.e. known) variables, $H = h_1, \ldots, h_l$ are the hidden (i.e. latent) variables, and each Q_i is the distribution of an individual latent variable h_i. The free parameters of this approximate model are the means μ_i of the distributions Q_i.

(Titov and Henderson, 2007) proposes two approximate models based on the variational approach. First, they show that the neural network of (Henderson, 2003) can be viewed as a coarse mean field approximation of ISBNs, which they call the feed-forward approximation. This approximation imposes the constraint that the free parameters μ_i of the approximate model are only allowed to depend on the distributions of their parent variables. This constraint increases the potential for a large approximation error, but it significantly simplifies the computations by allowing all the free parameters to be set in a single pass over the model.

The second approximation proposed in (Titov and Henderson, 2007) takes into consideration the fact that, after each decision is made, all the preceding latent variables should have their means μ_i updated. This approximation extends the feed-forward approximation to account for the most important components of this update. They call this approximation the mean field approximation, because a mean field approximation is applied to handle the statistical dependencies introduced by the new decisions. This approximation was shown to be a more accurate approximation of ISBNs than the feed-forward approximation, but remain tractable. It was also shown to achieve significantly better accuracy on constituent parsing.

2.4 Learning

Training these approximations of ISBNs is done to maximize the fit of the *approximate* models to the data. We use gradient descent, and a regularized maximum likelihood objective function. Gaussian regularization is applied, which is equivalent to the weight decay standardly used in neural networks. Regularization was reduced through the course of learning.

Gradient descent requires computing the derivatives of the objective function with respect to the model parameters. In the feed-forward approximation, this can be done with the standard Backpropagation learning used with neural networks. For the mean field approximation, propagating the error all the way back through the structure of the graphical model requires a more complicated calculation, but it can still be done efficiently (see (Titov and Henderson, 2007) for details).

3 The Dependency Parsing Algorithm

The sequences of decisions $D^1, ..., D^m$ which we will be modeling with ISBNs are the sequences of decisions made by a dependency parser. For this we use the parsing strategy for projective dependency parsing introduced in (Nivre et al., 2004), which is similar to a standard shift-reduce algorithm for context-free grammars (Aho et al., 1986). It can be viewed as a mixture of bottom-up and top-down parsing strategies, where left dependencies are constructed in a bottom-up fashion and right dependencies are constructed top-down. For details we refer the reader to (Nivre et al., 2004). In this section we briefly describe the algorithm and explain how we use it to define our history-based probability model.

In this paper, as in the CoNLL-X shared task, we consider labeled dependency parsing. The state of the parser is defined by the current stack S, the queue I of remaining input words and the partial labeled dependency structure constructed by previous parser decisions. The parser starts with an empty stack S and terminates when it reaches a configuration with an empty queue I. The algorithm uses 4 types of decisions:

1. The decision **Left-Arc**$_r$ adds a dependency arc from the next input word w_j to the word w_i on top of the stack and selects the label r for the

relation between w_i and w_j. Word w_i is then popped from the stack.

2. The decision **Right-Arc**$_r$ adds an arc from the word w_i on top of the stack to the next input word w_j and selects the label r for the relation between w_i and w_j.

3. The decision **Reduce** pops the word w_i from the stack.

4. The decision **Shift**$_{w_j}$ shifts the word w_j from the queue to the stack.

Unlike the original definition in (Nivre et al., 2004) the **Right-Arc**$_r$ decision does not shift w_j to the stack. However, the only thing the parser can do after a **Right-Arc**$_r$ decision is to choose the **Shift**$_{w_j}$ decision. This subtle modification does not change the actual parsing order, but it does simplify the definition of our graphical model, as explained in section 4.

We use a history-based probability model, which decomposes the probability of the parse according to the parser decisions:

$$P(T) = P(D^1, ..., D^m) = \prod_t P(D^t|D^1, \ldots, D^{t-1}),$$

where T is the parse tree and $D^1, \ldots, D^m$ is its equivalent sequence of parser decisions. Since we need a generative model, the action **Shift**$_{w_j}$ also predicts the next word in the queue I, w_{j+1}, thus the $P(Shift_{w_i}|D^1, \ldots, D^{t-1})$ is a probability both of the shift operation and the word w_{j+1} conditioned on current parsing history.[1]

Instead of treating each D^t as an atomic decision, it is convenient to split it into a sequence of elementary decisions $D^t = d_1^t, \ldots, d_n^t$:

$$P(D^t|D^1, \ldots, D^{t-1}) = \prod_k P(d_k^t|h(t,k)),$$

[1]In preliminary experiments, we also considered look-ahead, where the word is predicted earlier than it appears at the head of the queue I, and "anti-look-ahead", where the word is predicted only when it is shifted to the stack S. Early prediction allows conditioning decision probabilities on the words in the look-ahead and, thus, speeds up the search for an optimal decision sequence. However, the loss of accuracy with look-ahead was quite signifi cant. The described method, where a new word is predicted when it appears at the head of the queue, led to the most accurate model and quite effi cient search. The anti-look-ahead model was both less accurate and slower.

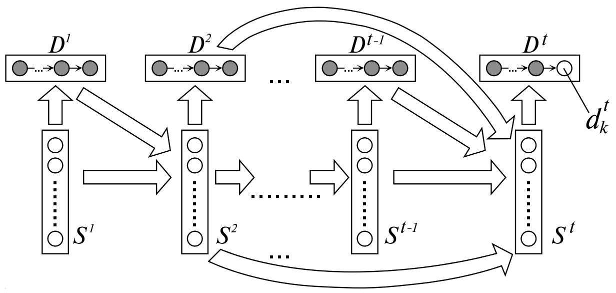


Figure 1: An ISBN for estimating $P(d_k^t|h(t,k))$.

where $h(t,k)$ denotes the parsing history $D^1, \ldots, D^{t-1}, d_1^t, \ldots, d_{k-1}^t$. We split **Left-Arc**$_r$ and **Right-Arc**$_r$ each into two elementary decisions: first, the parser decides to create the corresponding arc, then, it decides to assign a relation r to the arc. Similarly, we decompose the decision **Shift**$_{w_j}$ into an elementary decision to shift a word and a prediction of the word w_{j+1}. In our experiments we use datasets from the CoNLL-X shared task, which provide additional properties for each word token, such as its part-of-speech tag and some fine-grain features. This information implicitly induces word clustering, which we use in our model: first we predict a part-of-speech tag for the word, then a set of word features, treating feature combination as an atomic value, and only then a particular word form. This approach allows us to both decrease the effect of sparsity and to avoid normalization across all the words in the vocabulary, significantly reducing the computational expense of word prediction.

4 An ISBN for Dependency Parsing

In this section we define the ISBN model we use for dependency parsing. An example of this ISBN for estimating $P(d_k^t|h(t,k))$ is illustrated in figure 1. It is organized into vectors of variables: latent state variable vectors $S^{t'} = s_1^{t'}, \ldots, s_n^{t'}$, representing an intermediate state at position t', and decision variable vectors $D^{t'}$, representing a decision at position t', where $t' \leq t$. Variables whose value are given at the current decision (t,k) are shaded in figure 1, latent and current decision variables are left unshaded.

As illustrated by the edges in figure 1, the probability of each state variable $s_i^{t'}$ (the individual circles in $S^{t'}$) depends on all the variables in a finite set of relevant previous state and decision vectors, but there are no direct dependencies between the different variables in a single state vector. For each relevant decision vector, the precise set of decision variables which are connected in this way can be adapted to a particular language. As long as these connected decisions include all the new information about the parse, the performance of the model is not very sensitive to this choice. This is because ISBNs have the ability to induce their own complex features of the parse history, as demonstrated in the experiments in section 6.

The most important design decision in building an ISBN model is choosing the finite set of relevant previous state vectors for the current decision. By connecting to a previous state, we place that state in the local context of the current decision. This specification of the domain of locality determines the inductive bias of learning with ISBNs. When deciding what information to store in its latent variables, an ISBN is more likely to choose information which is immediately local to the current decision. This stored information then becomes local to any following connected decision, where it again has some chance of being chosen as relevant to that decision. In this way, the information available to a given decision can come from arbitrarily far away in the chain of interconnected states, but it is much more likely to come from a state which is relatively local. Thus, we need to choose the set of local (i.e. connected) states in accordance with our prior knowledge about which previous decisions are likely to be particularly relevant to the current decision.

To choose which previous decisions are particularly relevant to the current decision, we make use of the partial dependency structure which has been decided so far in the parse. Specifically, the current latent state vector is connected to a set of 7 previous latent state vectors (if they exist) according to the following relationships:

1. **Input Context**: the last previous state with the same queue I.
2. **Stack Context**: the last previous state with the same stack S.
3. **Right Child of Top of** S: the last previous state where the rightmost right child of the current stack top was on top of the stack.
4. **Left Child of Top of** S: the last previous state where the leftmost left child of the current stack top was on top of the stack.
5. **Left Child of Front of** I[2] : the last previous state where the leftmost child of the front element of I was on top of the stack.
6. **Head of Top**: the last previous state where the head word of the current stack top was on top of the stack.
7. **Top of** S **at Front of** I: the last previous state where the current stack top was at the front of the queue.

Each of these 7 relations has its own distinct weight matrix for the resulting edges in the ISBN, but the same weight matrix is used at each position where the relation is relevant.

All these relations but the last one are motivated by linguistic considerations. The current decision is primarily about what to do with the current word on the top of the stack and the current word on the front of the queue. The *Input Context* and *Stack Context* relationships connect to the most recent states used for making decisions about each of these words. The *Right Child of Top of* S relationship connects to a state used for making decisions about the most recently attached dependent of the stack top. Similarly, the *Left Child of Front of* I relationship connects to a state for the most recently attached dependent of the queue front. The *Left Child of Top of* S is the first dependent of the stack top, which is a particularly informative dependent for many languages. Likewise, the *Head of Top* can tell us a lot about the stack top, if it has been chosen already.

A second motivation for including a state in the local context of a decision is that it might contain information which has no other route for reaching the current decision. In particular, it is generally a good idea to ensure that the immediately preceding state is always included somewhere in the set of connected states. This requirement ensures that information, at least theoretically, can pass between any two states in the decision sequence, thereby avoiding any hard

[2]We refer to the *head* of the queue as the *front*, to avoid unnecessary ambiguity of the word *head* in the context of dependency parsing.

independence assumptions. The last relation, *Top of S at Front of I*, is included mainly to fulfill this requirement. Otherwise, after a **Shift**$_{w_j}$ operation, the preceding state would not be selected by any of the relationships.

As indicated in figure 1, the probability of each elementary decision $d_k^{t'}$ depends both on the current state vector $S^{t'}$ and on the previously chosen elementary action $d_{k-1}^{t'}$ from $D^{t'}$. This probability distribution has the form of a normalized exponential:

$$P(d_k^{t'} = d | S^{t'}, d_{k-1}^{t'}) = \frac{\Phi_{h(t',k)}(d)\, e^{\sum_j W_{dj} s_j^{t'}}}{\sum_{d'} \Phi_{h(t',k)}(d')\, e^{\sum_j W_{d'j} s_j^{t'}}},$$

where $\Phi_{h(t',k)}$ is the indicator function of the set of elementary decisions that may possibly follow the last decision in the history $h(t', k)$, and the W_{dj} are the weights. Now it is easy to see why the original decision **Right-Arc**$_r$ (Nivre et al., 2004) had to be decomposed into two distinct decisions: the decision to construct a labeled arc and the decision to shift the word. Use of this composite **Right-Arc**$_r$ would have required the introduction of individual parameters for each pair (w, r), where w is an arbitrary word in the lexicon and r - an arbitrary dependency relation.

5 Searching for the Best Tree

ISBNs define a probability model which does not make any a-priori assumptions of independence between any decision variables. As we discussed in section 4 use of relations based on partial output structure makes it possible to take into account statistical interdependencies between decisions closely related in the output structure, but separated by multiple decisions in the input structure. This property leads to exponential complexity of complete search. However, the success of the deterministic parsing strategy which uses the same parsing order (Nivre et al., 2006), suggests that it should be relatively easy to find an accurate approximation to the best parse with heuristic search methods. Unlike (Nivre et al., 2006), we can not use a lookahead in our generative model, as was discussed in section 3, so a greedy method is unlikely to lead to a good approximation. Instead we use a pruning strategy similar to that described in (Henderson, 2003), where it was applied to a considerably harder search problem: constituent parsing with a left-corner parsing order.

We apply fixed beam pruning after each decision **Shift**$_{w_j}$, because knowledge of the next word in the queue I helps distinguish unlikely decision sequences. We could have used best-first search between **Shift**$_{w_j}$ operations, but this still leads to relatively expensive computations, especially when the set of dependency relations is large. However, most of the word pairs can possibly participate only in a very limited number of distinct relations. Thus, we pursue only a fixed number of relations r after each **Left-Arc**$_r$ and **Right-Arc**$_r$ operation.

Experiments with a variety of post-shift beam widths confirmed that very small validation performance gains are achieved with widths larger than 30, and sometimes even a beam of 5 was sufficient. We found also that allowing 5 different relations after each dependency prediction operation was enough that it had virtually no effect on the validation accuracy.

6 Empirical Evaluation

In this section we evaluate the ISBN model for dependency parsing on three treebanks from the CoNLL-X shared task. We compare our generative models with the best parsers from the CoNLL-X task, including the SVM-based parser of (Nivre et al., 2006) (the MALT parser), which uses the same parsing algorithm. To test the feature induction abilities of our model we compare results with two feature sets, the feature set tuned individually for each language by (Nivre et al., 2006), and another feature set which includes only obvious local features. This simple feature set comprises only features of the word on top of the stack S and the front word of the queue I. We compare the gain from using tuned features with the similar gain obtained by the MALT parser. To obtain these results we train the MALT parser with the same two feature sets.[3]

In order to distinguish the contribution of ISBN's feature induction abilities from the contribution of

[3]The tuned feature sets were obtained from http://w3.msi.vxu.se/~nivre/research/MaltParser.html. We removed lookahead features for ISBN experiments but preserved them for experiments with the MALT parser. Analogously, we extended simple features with 3 words lookahead for the MALT parser experiments.

our estimation method and search, we perform another experiment. We use the tuned feature set and disable the feature induction abilities of the model by removing all the edges between latent variables vectors. Comparison of this restricted model with the full ISBN model shows how important the feature induction is. Also, comparison of this restricted model with the MALT parser, which uses the same set of features, indicates whether our generative estimation method and use of beam search is beneficial.

6.1 Experimental Setup

We used the CoNLL-X distributions of Danish DDT treebank (Kromann, 2003), Dutch Alpino treebank (van der Beek et al., 2002) and Slovene SDT treebank (Dzeroski et al., 2006). The choice of these treebanks was motivated by the fact that they all are freely distributed and have very different sizes of their training sets: 195,069 tokens for Dutch, 94,386 tokens for Danish and only 28,750 tokens for Slovene. As it is generally believed that discriminative models win over generative models with a large amount of training data, so we expected to see similar trend in our results. Test sets are about equal and contain about 5,000 scoring tokens.

We followed the experimental setup of the shared task and used all the information provided for the languages: gold standard part-of-speech tags and coarse part-of-speech tags, word form, word lemma (lemma information was not available for Danish) and a set of fine-grain word features. As we explained in section 3, we treated these sets of fine-grain features as an atomic value when predicting a word. However, when conditioning on words, we treated each component of this composite feature individually, as it proved to be useful on the development set. We used frequency cutoffs: we ignored any property (e.g., word form, feature or even part-of-speech tag[4]) which occurs in the training set less than 5 times. Following (Nivre et al., 2006), we used pseudo-projective transformation they proposed to cast non-projective parsing tasks as projective.

ISBN models were trained using a small development set taken out from the training set, which was used for tuning learning parameters and for early stopping. The sizes of the development sets were: 4,988 tokens for larger Dutch corpus, 2,504 tokens for Danish and 2,033 tokens for Slovene. The MALT parser was trained always using the entire training set. We expect that the mean field approximation should demonstrate better results than feed-forward approximation on this task as it is theoretically expected and confirmed on the constituent parsing task (Titov and Henderson, 2007). However, the sizes of testing sets would not allow us to perform any conclusive analysis, so we decided not to perform these comparisons here. Instead we used the mean field approximation for the smaller two corpora and used the feed-forward approximation for the larger one. Training the mean field approximations on the larger Dutch treebank is feasible, but would significantly reduce the possibilities for tuning the learning parameters on the development set and, thus, would increase the randomness of model comparisons.

All model selection was performed on the development set and a single model of each type was applied to the testing set. We used a state variable vector consisting of 80 binary variables, as it proved sufficient on the preliminary experiments. For the MALT parser we replicated the parameters from (Nivre et al., 2006) as described in detail on their web site.

The labeled attachment scores for the ISBN with tuned features (TF) and local features (LF) and ISBN with tuned features and no edges connecting latent variable vectors (TF-NA) are presented in table 1, along with results for the MALT parser both with tuned and local feature, the MST parser (McDonald et al., 2006), and the average score (Aver) across all systems in the CoNLL-X shared task. The MST parser is included because it demonstrated the best overall result in the task, non significantly outperforming the MALT parser, which, in turn, achieved the second best overall result. The labeled attachment score is computed using the same method as in the CoNLL-X shared task, i.e. ignoring punctuation. Note, that though we tried to completely replicate training of the MALT parser with the tuned features, we obtained slightly different results. The original published results for the MALT parser with tuned features were 84.8% for Danish, 78.6% for Dutch and 70.3% for Slovene. The im-

[4]Part-of-speech tags for multi-word units in the Danish treebank were formed as concatenation of tags of the words, which led to quite sparse set of part-of-speech tags.

		Danish	Dutch	Slovene
ISBN	TF	85.0	79.6	72.9
	LF	84.5	79.5	72.4
	TF-NA	83.5	76.4	71.7
MALT	TF	85.1	78.2	70.5
	LF	79.8	74.5	66.8
MST		84.8	79.2	73.4
Aver		78.3	70.7	65.2

Table 1: Labeled attachment score on the testing sets of Danish, Dutch and Slovene treebanks.

provement of the ISBN models (TF and LF) over the MALT parser is statistically significant for Dutch and Slovene. Differences between their results on Danish are not statistically significant.

6.2 Discussion of Results

The ISBN with tuned features (TF) achieved significantly better accuracy than the MALT parser on 2 languages (Dutch and Slovene), and demonstrated essentially the same accuracy on Danish. The results of the ISBN are among the two top published results on all three languages, including the best published results on Dutch. All three models, MST, MALT and ISBN, demonstrate much better results than the average result in the CoNLL-X shared task. These results suggest that our generative model is quite competitive with respect to the best models, which are both discriminative.[5] We would expect further improvement of ISBN results if we applied discriminative retraining (Henderson, 2004) or reranking with data-defined kernels (Henderson and Titov, 2005), even without introduction of any additional features.

We can see that the ISBN parser achieves about the same results with local features (LF). Local features by themselves are definitely not sufficient for the construction of accurate models, as seen from the results of the MALT parser with local features (and look-ahead). This result demonstrates that ISBNs are a powerful model for feature induction.

The results of the ISBN without edges connecting latent state vectors is slightly surprising and suggest that without feature induction the ISBN is significantly worse than the best models. This shows that

[5]Note that the development set accuracy predicted correctly the testing set ranking of ISBN TF, LF and TF-NA models on each of the datasets, so it is fair to compare the best ISBN result among the three with other parsers.

		to root	1	2	3 - 6	> 6
Da	ISBN	95.1	95.7	90.1	84.1	74.7
	MALT	95.4	96.0	90.8	84.0	71.6
Du	ISBN	79.8	92.4	86.2	81.4	71.1
	MALT	73.1	91.9	85.0	76.2	64.3
Sl	ISBN	76.1	92.5	85.6	79.6	54.3
	MALT	59.9	92.1	85.0	78.4	47.1
Av	ISBN	83.6	93.5	87.3	81.7	66.7
	MALT	76.2	93.3	87.0	79.5	61.0
	Improv	**7.5**	**0.2**	**0.4**	**2.2**	**5.7**

Table 2: F_1 score of labeled attachment as a function of dependency length on the testing sets of Danish, Dutch and Slovene.

the improvement is coming mostly from the ability of the ISBN to induce complex features and not from either using beam search or from the estimation procedure. It might also suggest that generative models are probably worse for the dependency parsing task than discriminative approaches (at least for larger datasets). This motivates further research into methods which combine powerful feature induction properties with the advantage of discriminative training. Although discriminative reranking of the generative model is likely to help, the derivation of fully discriminative feature induction methods is certainly more challenging.

In order to better understand differences in performance between ISBN and MALT, we analyzed how relation accuracy changes with the length of the head-dependent relation. The harmonic mean between precision and recall of labeled attachment, F_1 measure, for the ISBN and MALT parsers with tuned features is presented in table 2. F_1 score is computed for four different ranges of lengths and for attachments directly to root. Along with the results for each of the languages, the table includes their mean (Av) and the absolute improvement of the ISBN model over MALT (Improv). It is easy to see that accuracy of both models is generally similar for small distances (1 and 2), but as the distance grows the ISBN parser starts to significantly outperform MALT, achieving 5.7% average improvement on dependencies longer than 6 word tokens. When the MALT parser does not manage to recover a long dependency, the highest scoring action it can choose is to reduce the dependent from the stack without specifying its head, thereby attaching the dependent to the root by default. This explains the relatively low F_1 scores for attachments to root (evident for Dutch and Slovene): though recall of attachment to root is comparable to that of the ISBN parser (82.4% for MALT against 84.2% for ISBN, on average over 3 languages), precision for the MALT parser is much worse (71.5% for MALT against 83.1% for ISBN, on average).

The considerably worse accuracy of the MALT parser on longer dependencies might be explained both by use of a non-greedy search method in the ISBN and the ability of ISBNs to induce history features. To capture a long dependency, the MALT parser should keep a word on the stack during a long sequence of decision. If at any point during the intermediate steps this choice seems not to be locally optimal, then the MALT parser will choose the alternative and lose the possibility of the long dependency.[6] By using a beam search, the ISBN parser can maintain the possibility of the long dependency in its beam even when other alternatives seem locally preferable. Also, long dependences are often more difficult, and may be systematically different from local dependencies. The designer of a MALT parser needs to discover predictive features for long dependencies by hand, whereas the ISBN model can automatically discover them. Thus we expect that the feature induction abilities of ISBNs have a strong effect on the accuracy of long dependences. This prediction is confirmed by the differences between the results of the normal ISBN (TF) and the restricted ISBN (TF-NA) model. The TF-NA model, like the MALT parser, is biased toward attachment to root; it attaches to root 12.0% more words on average than the normal ISBN, without any improvement of recall and with a great loss of precision. The F_1 score on long dependences for the TF-NA model is also negatively effected in the same way as for the MALT parser. This confirms that the ability of the ISBN model to induce features is a major factor in improving accuracy of long dependencies.

[6]The MALT parser is trained to keep the word as long as possible: if both **Shift** and **Reduce** decisions are possible during training, it always prefers to shift. Though this strategy should generally reduce the described problem, it is evident from the low precision score for attachment to root, that it can not completely eliminate it.

7 Related Work

There has not been much previous work on latent variable models for dependency parsing. Dependency parsing with Dynamic Bayesian Networks was considered in (Peshkin and Savova, 2005), with limited success. Roughly, the model considered the whole sentence at a time, with the DBN being used to decide which words correspond to leaves of the tree. The chosen words are then removed from the sentence and the model is recursively applied to the reduced sentence. Recently several latent variable models for constituent parsing have been proposed (Koo and Collins, 2005; Matsuzaki et al., 2005; Prescher, 2005; Riezler et al., 2002). In (Matsuzaki et al., 2005) non-terminals in a standard PCFG model are augmented with latent variables. A similar model of (Prescher, 2005) uses a head-driven PCFG with latent heads, thus restricting the flexibility of the latent-variable model by using explicit linguistic constraints. While the model of (Matsuzaki et al., 2005) significantly outperforms the constrained model of (Prescher, 2005), they both are well below the state-of-the-art in constituent parsing. In (Koo and Collins, 2005), an undirected graphical model for constituent parse reranking uses dependency relations to define the edges. Thus, it should be easy to apply a similar method to reranking dependency trees.

Undirected graphical models, in particular Conditional Random Fields, are the standard tools for shallow parsing (Sha and Pereira, 2003). However, shallow parsing is effectively a sequence labeling problem and therefore differs significantly from full parsing. As discussed in (Titov and Henderson, 2007), undirected graphical models do not seem to be suitable for history-based parsing models.

Sigmoid Belief Networks (SBNs) were used originally for character recognition tasks, but later a dynamic modification of this model was applied to the reinforcement learning task (Sallans, 2002). However, their graphical model, approximation method, and learning method differ significantly from those of this paper. The extension of dynamic SBNs with incrementally specified model structure (i.e. Incremental Sigmoid Belief Networks, used in this paper) was proposed and applied to constituent parsing in (Titov and Henderson, 2007).

8 Conclusions

We proposed a latent variable dependency parsing model based on Incremental Sigmoid Belief Networks. Unlike state-of-the-art dependency parsers, it uses a generative history-based model. We demonstrated that it achieves state-of-the-art results on a selection of languages from the CoNLL-X shared task. The parser uses a vector of latent variables to represent an intermediate state and uses relations defined on the output structure to construct the edges between latent state vectors. These properties make it a powerful feature induction method for dependency parsing, and it achieves competitive results even with very simple explicit features. The ISBN model is especially accurate at modeling long dependences, achieving average improvement of 5.7% over the state-of-the-art baseline on dependences longer than 6 words. Empirical evaluation demonstrates that competitive results are achieved mostly because of the ability of the model to induce complex features and not because of the use of a generative probability model or a specific search method. As with other generative models, it can be further improved by the application of discriminative reranking techniques. Discriminative methods are likely to allow it to significantly improve over the current state-of-the-art in dependency parsing.[7]

Acknowledgments

This work was funded by Swiss NSF grant 200020-109685, UK EPSRC grant EP/E019501/1, and EU FP6 grant 507802 for project TALK. We thank Joakim Nivre and Sandra Kübler for an excellent tutorial on dependency parsing given at COLING-ACL 2006.

References

Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. 1986. *Compilers: Principles, Techniques and Tools*. Addison Wesley.

Leon Bottou. 1991. *Une approche théoretique de l'apprentissage connexionniste: Applications à la reconnaissance de la parole*. Ph.D. thesis, Université de Paris XI, Paris, France.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proc. of the Tenth Conference on Computational Natural Language Learning*, New York, USA.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fi ne n-best parsing and MaxEnt discriminative reranking. In *Proc. 43rd Meeting of Association for Computational Linguistics*, pages 173–180, Ann Arbor, MI.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proc. 1st Meeting of North American Chapter of Association for Computational Linguistics*, pages 132–139, Seattle, Washington.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proc. 17th Int. Conf. on Machine Learning*, pages 175–182, Stanford, CA.

S. Dzeroski, T. Erjavec, N. Ledinek, P. Pajas, Z. Zabokrtsky, and A. Zele. 2006. Towards a Slovene dependency treebank. In *Proc. Int. Conf. on Language Resources and Evaluation (LREC)*, Genoa, Italy.

James Henderson and Ivan Titov. 2005. Data-defi ned kernels for parse reranking derived from probabilistic models. In *Proc. 43rd Meeting of Association for Computational Linguistics*, Ann Arbor, MI.

James Henderson. 2003. Inducing history representations for broad coverage statistical parsing. In *Proc. joint meeting of North American Chapter of the Association for Computational Linguistics and the Human Language Technology Conf.*, pages 103–110, Edmonton, Canada.

James Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proc. 42nd Meeting of Association for Computational Linguistics*, Barcelona, Spain.

M. I. Jordan, Z.Ghahramani, T. S. Jaakkola, and L. K. Saul. 1999. An introduction to variational methods for graphical models. In Michael I. Jordan, editor, *Learning in Graphical Models*. MIT Press, Cambridge, MA.

Terry Koo and Michael Collins. 2005. Hidden-variable models for discriminative reranking. In *Proc. Conf. on Empirical Methods in Natural Language Processing*, Vancouver, B.C., Canada.

Matthias T. Kromann. 2003. The Danish dependency treebank and the underlying linguistic theory. In *Proceedings of the 2nd Workshop on Treebanks and Linguistic Theories (TLT)*, Vaxjo, Sweden.

[7]The ISBN dependency parser will be soon made downloadable from the authors' web-page.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proceedings of the 43rd Annual Meeting of the ACL*, Ann Arbor, MI.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proc. of the Tenth Conference on Computational Natural Language Learning*, New York, USA.

Kevin P. Murphy. 2002. *Dynamic Belief Networks: Representation, Inference and Learning*. Ph.D. thesis, University of California, Berkeley, CA.

Radford Neal. 1992. Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In *Proc. of the Eighth Conference on Computational Natural Language Learning*, pages 49–56, Boston, USA.

Joakim Nivre, Johan Hall, Jens Nilsson, Gulsen Eryigit, and Svetoslav Marinov. 2006. Pseudo-projective dependency parsing with support vector machines. In *Proc. of the Tenth Conference on Computational Natural Language Learning*, pages 221–225, New York, USA.

Leon Peshkin and Virginia Savova. 2005. Dependency parsing with dynamic Bayesian network. In *AAAI, 20th National Conference on Artificial Intelligence*, Pittsburgh, Pennsylvania.

Detlef Prescher. 2005. Head-driven PCFGs with latent-head statistics. In *Proc. 9th Int. Workshop on Parsing Technologies*, Vancouver, Canada.

Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proc. 40th Meeting of Association for Computational Linguistics*, Philadelphia, PA.

Brian Sallans. 2002. *Reinforcement Learning for Factored Markov Decision Processes*. Ph.D. thesis, University of Toronto, Toronto, Canada.

Fei Sha and Fernando Pereira. 2003. Shallow parsing with conditional random fi elds. In *Proc. joint meeting of North American Chapter of the Association for Computational Linguistics and the Human Language Technology Conf.*, Edmonton, Canada.

Ivan Titov and James Henderson. 2007. Constituent parsing with incremental sigmoid belief networks. In *Proc. 45th Meeting of Association for Computational Linguistics*, Prague, Czech Republic.

L. van der Beek, G. Bouma, J. Daciuk, T. Gaustad, R. Malouf, G van Noord, R. Prins, and B. Villada. 2002. The Alpino dependency treebank. *Computational Linguistic in the Netherlands (CLIN)*.

Three-Dimensional Parametrization for Parsing Morphologically Rich Languages

Reut Tsarfaty and Khalil Sima'an
Institute for Logic, Language and Computation
University of Amsterdam
Plantage Muidergracht 24, 1018TV Amsterdam, The Netherlands
{rtsarfat,simaan}@science.uva.nl

Abstract

Current parameters of accurate unlexicalized parsers based on Probabilistic Context-Free Grammars (PCFGs) form a two-dimensional grid in which rewrite events are conditioned on both horizontal (head-outward) and vertical (parental) histories. In Semitic languages, where arguments may move around rather freely and phrase-structures are often shallow, there are additional morphological factors that govern the generation process. Here we propose that agreement features percolated up the parse-tree form a third dimension of parametrization that is orthogonal to the previous two. This dimension differs from mere "state-splits" as it applies to a whole set of categories rather than to individual ones and encodes linguistically motivated co-occurrences between them. This paper presents extensive experiments with extensions of unlexicalized PCFGs for parsing Modern Hebrew in which tuning the parameters in three dimensions gradually leads to improved performance. Our best result introduces a new, stronger, lower bound on the performance of treebank grammars for parsing Modern Hebrew, and is on a par with current results for parsing Modern Standard Arabic obtained by a fully lexicalized parser trained on a much larger treebank.

1 Dimensions of Unlexicalized Parsing

Probabilistic Context Free Grammars (PCFGs) are the formal backbone of most high-accuracy statistical parsers for English, and a variety of techniques was developed to enhance their performance relative to the naïve treebank implementation — from unlexicalized extensions exploiting simple category splits (Johnson, 1998; Klein and Manning, 2003) to fully lexicalized parsers that condition events below a constituent upon the head and additional lexical content (Collins, 2003; Charniak, 1997). While it is clear that conditioning on lexical content improves the grammar's disambiguation capabilities, Klein and Manning (2003) demonstrate that a well-crafted unlexicalized PCFG can close the gap, to a large extent, with current state-of-the-art lexicalized parsers for English.

The factor that sets apart vanilla PCFGs (Charniak, 1996) from their unlexicalized extensions proposed by, e.g., (Johnson, 1998; Klein and Manning, 2003), is the choice for statistical parametrization that weakens the independence assumptions implicit in the treebank grammar. Studies on accurate unlexicalized parsing models outline two dimensions of parametrization. The first, proposed by (Johnson, 1998), is the annotation of parental history, and the second encodes a head-outward generation process (Collins, 2003). Johnson (1998) augments node labels with the label of their parent, thus incorporating a dependency on the node's grandparent. Collins (2003) proposes to generate the head of a phrase first and then generate its sisters using Markovian processes, thereby exploiting head/sister-dependencies.

Klein and Manning (2003) systematize the distinction between these two forms of parametrization by drawing them on a horizontal-vertical grid: parent encoding is vertical (external to the rule) whereas head-outward generation is horizontal (internal to the rule). By varying the value of the parameters along the grid, Klein and Manning (2003) tune their treebank grammar to achieve improved performance. This two-dimensional parametrization has been instrumental in devising parsing models that improve disambiguation capabilities for English as well as other languages, such as German (Dubey and Keller, 2003) Czech (Collins et al., 1999) and Chinese (Bikel and Chiang, 2000). However, accuracy results for parsing languages other than English still lag behind.[1]

We propose that for various languages including the Semitic family, e.g. Modern Hebrew (MH) and Modern Standard Arabic (MSA), a third dimension of parametrization is necessary for encoding linguistic information relevant for breaking false independence assumptions. In Semitic languages, arguments may move around rather freely and the phrase-structure of clause-level categories is often shallow. For such languages agreement features play a role in disambiguation at least as important as the vertical and horizontal conditioning. We propose a third dimension of parameterizations that encodes morphological features such as those realizing syntactic agreement. These features are percolated from surface forms in a bottom-up fashion and express information that is complementary to the horizontal and vertical generation histories proposed before. Such morphological information refines syntactic categories based on their morpho-syntactic role, and captures linguistically motivated co-occurrences and dependencies manifested via, e.g., morpho-syntactic agreement.

This work aims at parsing MH and explores the empirical contribution of the three dimensions of parameters specified above. We present extensive experiments that gradually lead to improved performance as we extend the degree to which the three dimensions are exploited. Our best model uses all three dimensions of parametrization, and our best result is on a par with those achieved for MSA using a fully lexicalized parser and a much larger treebank. The remainder of this document is organized as follows. In section 2 we review characteristic aspects of MH (and other Semitic languages) and illustrate the special role of morphology and dependencies displayed by morpho-syntactic processes using the case of syntactic definiteness in MH. In section 3 we define our three-dimensional parametrization space. In section 4 we spell out the method and procedure for the empirical evaluation of one, two and three parametrization dimensions, and in section 5 we report and analyze results for different parametrization choices. Finally, section 6 discusses related work and in section 7 we summarize and conclude.

2 Dimensions of Modern Hebrew Syntax

Parsing MH is in its infancy. Although a syntactically annotated corpus has been available for quite some time (Sima'an et al., 2001), we know of only two studies attempting to parse MH using statistical methods (see section 6). One reason for the sparseness in this field is that the adaptation of existing models to parsing MH is technically involved yet does not guarantee to yield comparable results as the processes that license grammatical structures of phrases and sentences in MH differ from those assumed for English. This section outlines differences between English and MH and discusses their reflection in the MH treebank annotation scheme. We argue that on top of syntactic processes exploited by current parsers there is an orthogonal morpho-syntactic dimension which is invaluable for syntactic disambiguation, and it can be effectively learned using simple treebank grammars.

2.1 Modern Hebrew Structure

Phrases and sentences in MH, as well as in Arabic and other Semitic languages, have a relatively flexible phrase structure. Subjects, verbs and objects can be inverted and prepositional phrases, adjuncts and verbal modifiers can move around rather freely. The factors that affect word-order in the language are not exclusively syntactic and have to do with rhetorical and pragmatic factors as well.[2]

[1]The learning curves over increasing training data (e.g., for German (Dubey and Keller, 2003)) show that treebank size cannot be the sole factor to account for the inferior performance.

[2]See, for instance, (Melnik, 2002) for an Information Structure-syntactic account of verb initial sentences.

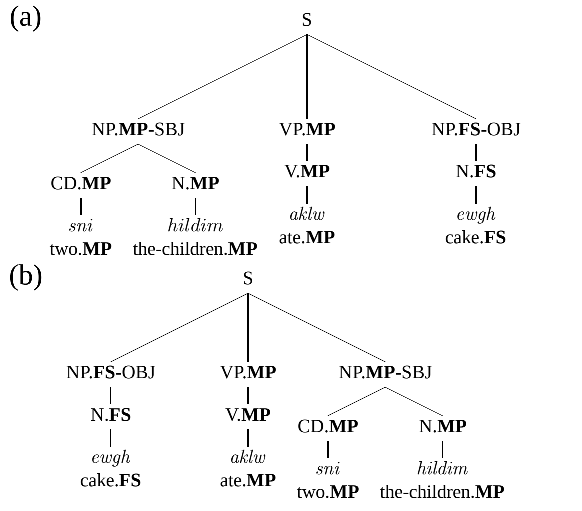


Figure 1: **Word Order and Agreement Features in MH Phrases:** Agreement on **MP** features reveals the subject-predicate dependency between surface forms and their dominating constituents in a variable phrase-structure (marking **M**(asculine), **F**(eminine), **S**(ingular), **P**(lural).)

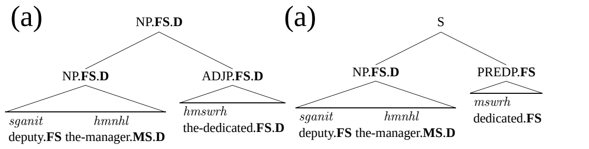


Figure 2: **Definiteness in MH as a Phrase-Level Agreement Feature:** Agreement on definiteness helps to determine the internal structure of a higher level NP (a), and the absence thereof helps to determine the attachment to a predicate in a verb-less sentence (b) (marking **D**(efiniteness))

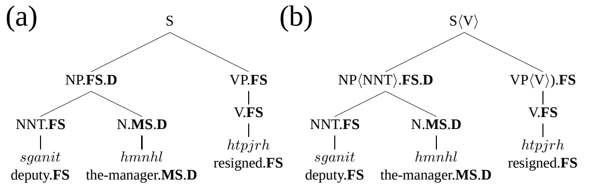


Figure 3: **Phrase-Level Agreement Features and Head-Dependencies in MH:** The direction of percolating definiteness in MH is distinct of that of the head (marking ⟨head-tag⟩)

It would be too strong a claim, however, to classify MH (and similar languages) as a free-word-order language in the canonical sense. The level of freedom in the order and number of internal constituents varies between syntactic categories. Within a verb phrase or a sentential clause, for instance, the order of constituents obeys less strict rules than within, e.g., a noun phrase.[3] Figure 1 illustrates two syntactic structures that express the same grammatical relations yet vary in their internal order of constituents. Within the noun phrase constituents, however, determiners always precede nouns.

Within the flexible phrase structure it is typically morphological information that provides cues for the grammatical relations between surface forms. In figure 1, for example, it is agreement on gender and number that reveals the subject-predicate dependency between surface forms. Figure 1 also shows that agreement features help to reveal such relations between higher levels of constituents as well.

Determining the child constituents that contribute each of the features is not a trivial matter either. To illustrate the extent and the complexity of that matter let us consider *definiteness* in MH, which is morphologically marked (as an h prefix to the stem, glossed here explicitly as "the-") and behaves as a syntactic property (Danon, 2001). Definite noun-phrases exhibit agreement with other modifying phrases, and such agreement helps to determine the internal structure, labels, and the correct level of attachment as illustrated in figure 2. The agreement on definiteness helps to determine the internal structure of noun phrases 2(a), and the absence thereof helps in determining the attachment to predicates in verb-less sentences, as in 2(b). Finally, definiteness may be percolated from a different form than the one determining the gender and number of a phrase. In figure 3(a), for instance, the definiteness feature (marked as D) percolates from '$hmnhl$' (the-manager.MS.D) while the gender and number are percolated from '$sganit$' (deputy.FS). The direction of percolation of definiteness may be distinct of that of percolating head information, as can be seem in figure 3(b). (The direction of head-dependencies in MH typically coincides with that of percolating gender.)

To summarize, agreement features are helpful in analyzing and disambiguating syntactic structures in MH, not only at the lexical level, but also at higher levels of constituency. In MH, features percolated from different surface forms jointly determine the features of higher-level constituents, and such features manifest multiple dependencies, which in turn cannot be collapsed onto a single head.

[3]See (Wintner, 2000) and (Goldberg et al., 2006) for formal and statistical accounts (respectively) of noun phrases in MH.

2.2 The Modern Hebrew Treebank Scheme

The annotation scheme of version 2.0 of the MH treebank (Sima'an et al., 2001)[4] aims to capture the morphological and syntactic properties of MH just described. This results in several aspects that distinguish the MH treebank from, e.g., the WSJ Penn treebank annotation scheme (Marcus et al., 1994).

The MH treebank is built over word segments. This means that the yields of the syntactic trees do not correspond to space delimited words but rather to morphological segments that carry distinct syntactic roles, i.e., each segment corresponds to a single POS tag. (This in turn means that prefixes marking determiners, relativizers, prepositions and definite articles are segmented away and appear as leaves in a syntactic parse tree.) The POS categories assigned to segmented words are decorated with features such as gender, number, person and tense, and these features are percolated higher up the tree according to pre-defined syntactic dependencies (Krymolowski et al., 2007). Since agreement features of non-terminal constituents may be contributed by more than one child, the annotation scheme defines multiple dependency labels that guide the percolation of the different features higher up the tree. Definiteness in the MH treebank is treated as a segment at the POS tags level and as a feature at the level of non-terminals. As any other feature, it is percolated higher up the tree according to marked dependency labels. Table 1 lists the features and values annotated on top of syntactic categories and table 2 describes the dependencies according to which these features are percolated from child constituents to their parents.

In order to comply with the flexible phrase structure in MH, clausal categories (S, SBAR and FRAG and their corresponding interrogatives SQ, SQBAR and FRAGQ) are annotated as flat structures. Verbs (VB tags) always attach to a VP mother, however only non-finite VBs can accept complements under the same VP parent, meaning that all inflected verb forms are represented as unary productions under an inflected VP. NP and PP are annotated as nested structures capturing the recursive structure of construct-state nouns, numerical expressions and possession. An additional category, PREDP, is added in the treebank scheme to account for sentences in MH that lack a copular element, and it may also be decorated with inflectional features agreeing with the subject. The MH treebank scheme also features null elements that mark traces and additional labels that mark functional features (e.g., SBJ,OBJ) which we strip off and ignore throughout this study.

Feature:Value	**Value Encoded**
gender:Z	masculine
gender:N	feminine
gender:B	both
number:Y	singular
number:R	plural
number:B	both
definiteness:H	definite
definiteness:U	underspecified

Table 1: **Features and Values in the MH Treebank**

Dependency Type	**Features Percolated**
DEP_HEAD	all
DEP_MAJOR	at least gender
DEP_NUMBER	number
DEP_DEFINITE	definiteness
DEP_ACCUSATIVE	case
DEP_MULTIPLE	all (e.g., conjunction)

Table 2: **Dependency Labels in the MH Treebank**

Morphological features percolated up the tree manifest dependencies that are marked locally yet have a global effect. We propose to learn treebank grammars in which the syntactic categories are augmented with morphological features at all levels of the hierarchy. This allows to learn finer-grained categories with subtle differences in their syntactic behavior and to capture non-independence between certain parts of the syntactic parse-tree.

3 Refining the Parameter Space

(Klein and Manning, 2003) argue that parent encoding on top of syntactic categories and RHS markovization of CFG productions are two instances of the same idea, namely that of encoding the generation history of a node to a varying degree. They subsequently describe two dimensions that define their parameters' space. The *vertical* dimension (v), capturing the history of the node's ancestors in a top-

[4]Version 2.0 of the MH treebank is publicly available at `http://mila.cs.technion.ac.il/english/index.html` along with a complete overview of the MH annotation scheme and illustrative examples (Krymolowski et al., 2007).

down generation process (e.g., its parent and grandparent), and the *horizontal* dimension (h), capturing the previously generated horizontal ancestors of a node (effectively, its sisters) in a head-outward generation process. By varying the value of h and v along this two-dimensional grid they improve performance of their induced treebank grammar.

Formally, the probability of a parse tree π is calculated as the probability of its derivation, the sequential application of rewrite rules. This in turn is calculated as the product of rules' probabilities, approximated by assuming independence between them $P(\pi) = \prod_i P(r_i | r_1 \circ ... \circ r_{i-1}) \approx \prod_i P(r_i)$. The vertical dimension v can be thought of as a function Ψ_0 selecting features from the generation history of the constituent thus restoring selected dependencies:

$$P(r_i) = P(r_i | \Psi_0(r_1 \circ .. \circ r_{i-1}))$$

The horizontal dimension h can be thought of as two functions Ψ_1, Ψ_2 over decomposed rules, where Ψ_1 selects hidden internal features of the parent, and Ψ_2 selects previously generated sisters in a head-outward Markovian process (we retain here the assumption that the head child H always matters).

$$P(r_i) = P_h(H | \Psi_1(LHS(r_i))) \times \prod_{C \in RHS(r_i) - H} P_C(C | \Psi_2(RHS(r_i)), H)$$

The fact that the default notion of a treebank grammar takes $v = 1$ (i.e., $\Psi_0(r_1 \circ .. \circ r_{i-1}) = \emptyset$) and $h = \infty$ (RHS cannot decompose) is, according to Klein and Manning (2003), a historical accident.

We claim that languages with freeer word order and richer morphology call for an additional dimension of parametrization. The additional parameter shows to what extent morphological features encoded in a specialized structure back up the derivation of the tree. This dimension can be thought of as a function Ψ_3 selecting aspects of morphological orthogonal analysis of the rules, where MA denotes morphological analysis of the syntactic categories in both LHS and RHS of the rule.

$$P(r_i) = P(r_i | \Psi_3(MA(r_i)))$$

The fact that in current parsers $\Phi_3(MA(r_i)) = \emptyset$ is, we claim, another historical accident. Parsing English is quite remarkable in that it can be done with

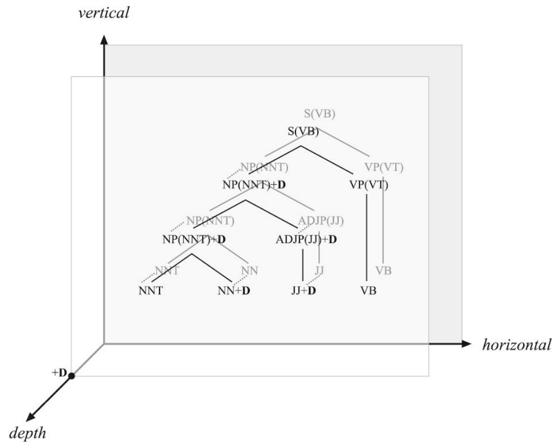
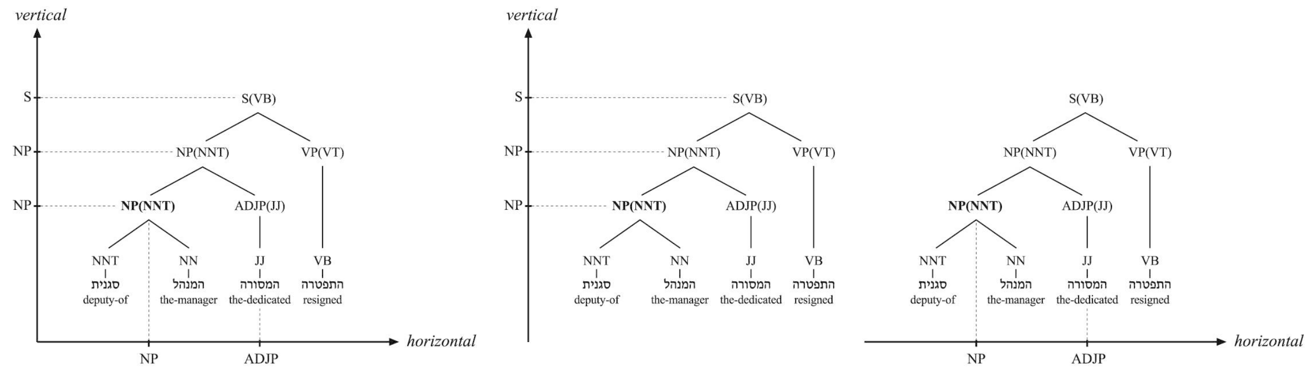


Figure 4: **The Three-Dimensional Parametrization Space**

impoverished morphological treatment, but for languages in which morphological processes are more pertinent, we argue, bi-dimensional parametrization shall not suffice.

The emerging picture is as follows. Bare-category skeletons reside in a bi-dimensional parametrization space (figure 3(a)) in which the vertical (figure 3(b)) and horizontal (figure 3(c)) parameter instantiations elaborate the generation history of a non-terminal node. Specialized structures enriched with (an increasing amount of) morphological features reside deeper along a third dimension we refer to as *depth* (d). Figure 4 illustrates an instantiation of $d = 1$ with a single definiteness feature. Higher d values would imply adding more (accumulating) features.

Klein and Manning (2003) view the *vertical* and *horizontal* parametrization dimensions as implementing *external* and *internal* annotation strategies respectively. External parameters indicate features of the external environment that influence the node's expansion possibilities, and internal parameters mark aspects of hidden internal content which influence constituents' external distribution. We view the third dimension of parametrization as implementing a *relational* strategy of annotation encoding the way different constituents may combine to form phrases and sentences. In a bottom up process this annotation strategy imposes soft constraints on a the top-down head-outward generation process. Figure 6(a) focuses on a selected NP node highlighted in figure 4 and shows its expansion possibilities in three dimensions. Figure 6(b) illustrates how the depth expansion interacts with both parent anno-



(a) **The** *horizontal/vertical* **Grid** (b) **The** *vertical* **dimension** (c) **The** *horizontal* **dimension**

Figure 5: **The Two-Dimensional Space:** The horizontal and vertical dimensions outlined by (Klein and Manning, 2003)

tation and neighbor dependencies thereby affecting both distributions.

3.1 A Note on State-Splits

Recent studies (Klein and Manning, 2003; Matsuzaki et al., 2005; Prescher, 2005; Petrov et al., 2006) suggest that category-splits help in enhancing the performance of treebank grammars, and a previous study on MH (Tsarfaty, 2006) outlines specific POS-tags splits that improve MH parsing accuracy. Yet, there is a major difference between category-splits, whether manually or automatically acquired, and the kind of state-splits that arise from agreement features that refine phrasal categories. While category-splits aim at each category in isolation, agreement features apply to a whole set of categories all at once, thereby capturing refinement of the categories as well as linguistically motivated co-occurrences between them. Individual category-splits are viewed as taking place in a two-dimensional space and it is hard to analyze and empirically evaluate their interaction with other annotation strategies. Here we propose a principled way to statistically model the interaction between different linguistic processes that license grammatical structures and empirically contrast their contribution.

3.2 A Note on Stochastic AV grammars

The practice of having morphological features orthogonal to a constituency structure is not a new one and is familiar from formal theories of syntax such as HPSG (Sag et al., 2003) and LFG (Kaplan and Bresnan, 1982). Here we propose to reframe systematic morphological decoration of syntactic categories at all levels of the hierarchy as

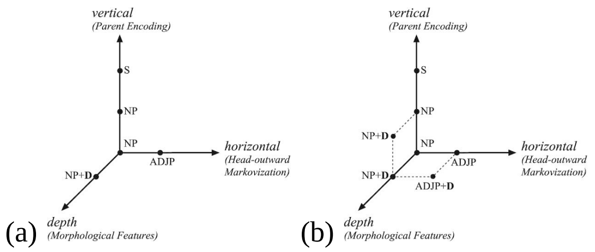


Figure 6: **The Expansion Possibilities of a Non-Terminal Node:** Expanding the NP from figure 4 in a three-dimensional parameterization Space

an additional dimension of statistical estimation for learning unlexicalized treebank PCFGs. Our proposal deviates from various stochastic extensions of such constraints-based grammatical formalisms (cf. (Abney, 1997)) and has the advantage of elegantly bypassing the issue of loosing probability mass to failed derivations due to unification failures. To the best of our knowledge, this proposal has not been empirically explored before.

4 Experimental Setup

Our goal is to determine the optimal strategy for learning treebank grammars for MH and to contrast it with bi-dimensional strategies explored for English. The methodology we use is adopted from (Klein and Manning, 2003) and our procedure is identical to the one described in (Johnson, 1998). We define transformations over the treebank that accept as input specific points in the (h, v, d) space depicted in figure 7. We use the transformed training sets for learning different treebank PCFGs which we then used to parse unseen sentences, and detransform the parses for the purpose of evaluation.[5]

[5]Previous studied on MH used different portions of the treebank and its annotation scheme due to its gradual development

Data We use version 2.0 of the MH treebank which consists of 6501 sentences from the daily newspaper 'Ha'aretz'. We employ the syntactic categories, POS categories and morphological features annotated therein. The data set is split into 13 sections consisting of 500 sentences each. We use the first section (section 0) as our development set and the last section (section 12) as our test set. The remaining sentences (sections 1–11) are all used for training. After removing empty sentences, sentences with uneven bracketing and sentences that do not match the annotation scheme[6] we remain with a *devset* of 483 sentences (average length in word segments 48), a *trainset* of 5241 sentences (53) and a *testset* of 496 sentences (58). Since this work is only the first step towards the development of a broad-coverage statistical parser for MH (and other Semitic languages) we use the development set for parameter-tuning and error analysis and use the test set only for confirming our best results.

Models The models we implement use one-, two- or three-dimensional parametrization and different instantiation of values thereof. (Due to the small size of our data set we only use the values $\{0, 1\}$ as possible instantiations.)

The v dimension is implemented using a transform as in (Johnson, 1998) where $v = 0$ corresponds to bare syntactic categories and $v = 1$ augments node labels with the label of their parent node.

The h dimension is peculiar in that it distinguishes PCFGs ($h = \infty$), where RHS cannot decompose, from their head-driven unlexicalized variety. To implement $h \neq \infty$ we use a PCFG transformation emulating (Collins, 2003)'s first model, in which sisters are generated conditioned on the head tag and a simple 'distance' function (Hageloh, 2007).[7] The instantiated value of h then selects the number of previously generated (non-head) sisters to be taken into account when generating the next sister in a Markovian process (Ψ_2 in our formal exposition).

The d dimension we proposed is implemented using a transformation that augments syntactic categories with morphological features percolated up the tree. We use $d = 0$ to select bare syntactic categories and instantiate $d = 1$ with the definiteness feature. The decision to select definiteness (rather than, e.g., gender or number) is rather pragmatic as its direction of percolation may be distinct of head information and the question remains whether the combination of such non-overlapping dependencies is instrumental for parsing MH.

Our baseline model is a vanilla treebank PCFG as described in (Charniak, 1996) which we locate on the $(\infty, 0, 0)$ point of our coordinates-system. In a first set of experiments we implement simple PCFG extensions of the treebank trees based on selected points on the (∞, v, d) plain. In a second set of experiments we use an unlexicalized head-driven baseline à la (Collins, 2003) located on the $(0, 0, 0)$ coordinate. We transform the treebank trees in correspondence with different points in the three-dimensional space defined by (h, v, d). The models we implement are marked in the coordinate-system depicted in figure 7. The implementation details of the transformations we use are spelled out in tables 3–4.

Procedure We implement different models that correspond to different instantiations of h, v and d. For each instantiation we transform the training set and learn a PCFG using Maximum Likelihood estimates, and we use BitPar (Schmidt, 2004), an efficient general-purpose parser, to parse unseen sentences. The input to the parser is a sequence of word segments where each segment corresponds to a single POS tag, possibly decorated with morphological features. This setup assumes partial morphological disambiguation (namely, segmentation) but crucially we do *not* disambiguate their respective POS categories. This setup is more appropriate for using general-purpose parsing tools and it makes our results comparable to studies in other languages.[8]

process. As the MH treebank is approaching maturity we feel that the time is ripe to standardize its use for MH statistical parsing. The software we implemented will be made available for non-commercial use upon request to the author(s) and the feature percolation software by (Krymolowski et al., 2007) is publicly available through the Knowledge Center for Processing Hebrew. By this we hope to increase the interest in MH within the parsing community and to facilitate the application of more sophisticated models by cutting down on setup time.

[6]Marked as "NO_MATCH" in the treebank.

[7]A formal overview of the transformation and its correspondence to (Collins, 2003)'s models is available at (Hageloh, 2007). We use the distance function defined therein, marking the direction and whether it is the first node to be generated.

[8]Our working assumption is that better performance of a parsing model in our setup will improve performance also

Transliterate The lexical items (leaves) in the MH treebank are written left-to-write and are encoded in utf8. A transliteration software is used to convert the utf encoding into Latin characters and to reverse their order, essentially allowing for standard left-to-right processing.
Correct The manual annotation resulted in unavoidable errors in the annotation scheme, such as typos (e.g., SQBQR instead of SQBAR) wrong delimiters (e.g., "-" instead of "_") or wrong feature order (e.g., number-gender instead of gender-number). We used an automatic script to detect these error, we manually determine their correction. Then we created an automatic script to apply all fixes (57 errors in 1% sentences).
Re-attach VB elements are attached by convention to a VP which inherits its morphological features. 9 VB instances in the treebank are mistakenly attached to an S parent without an intermediate VP level. Our software re-attaches those VB elements to a VP parent and percolates its morphological features.
Disjoint Due to recursive processes of generating noun phrases and numerical expression (*smixut)* in MH the sets of POS and syntactic categories are not disjoint. This is a major concern for PCFG parsers that assume disjoint sets of pre- and non-terminals. The overlap between the sets also introduces additional infinite derivations to which we loose probability mass. Our software takes care to decorate POS categories used as non-terminal with an additional "P", creating a new set of categories encoding partial derivations.
Lexicalize A pre-condition for applying horizontal parameterizations à la Collins is the annotation of heads of syntactic phrases. The treebank provided by the knowledge center does not define unique heads, but rather, mark multiple dependencies for some categories and none for others. Our software uses rules for choosing the syntactic head according to specified dependencies and a head table when none are specified.
Linearize In order to implement the head-outward constituents' generation process we use software made available to us by (Hageloh, 2007) which converts PCFG production such as the generation of a head is followed by left and right markovized derivation processes. We used two versions of Markovization, one which conditions only on the head and a distance function, and another which conditions also on immediately neighboring sister(s).
Decorate Our software implements an additional general transform which selects the features that are to be annotated on top of syntactic categories to implement various parametrization decisions. This transform can be used for, e.g., displaying parent information, selecting morphological features, etc.

Table 3: **Transforms over the MH Treebank:** We clean and correct the treebank using **Transliterate, Correct, Re-attach** and **Disjoint**, and transform the training set according to certain parametrization decisions using **Lexicalize, Linearize** and **Decorate**.

Smoothing pre-terminal rules is done explicitly by collecting statistics on "rare word" occurrences and providing the parser with possible open class categories and their corresponding frequency counts. The frequency threshold defining "rare words" was tuned empirically and set to 1. The resulting test parses are detransformed and to skeletal constituent structures, and are compared against the gold parses to evaluate parsing accuracy.

Evaluation We evaluate our models using EVALB in accordance with standard PARSEVAL evaluation metrics. The evaluation of all models focuses on Labeled Precision and Recall considering bare syntactic categories (stripping off all morphological or parental features and removing intermediate nodes for linearization). We report the average F-measure for sentences of length up to 40 and for all sentences ($F_{\leq 40}$ and F_{All} respectively). We report the results for two evaluation options, once including punctuation marks (WP) and once excluding them (WOP).

within an integrated model for morphological and syntactic disambiguation in the spirit of (Tsarfaty, 2006). We conjecture that the kind of models developed here which takes into account morphological information is more appropriate for the morphological disambiguation task defined therein.

5 Results

Our baseline for the first set of experiments is a vanilla PCFG as described in (Charniak, 1996) (without a preceding POS tagging phase and without right branching corrections). We transform the treebank trees based on various points in the (∞, v, d) two-dimensional space to evaluate the performance of the resulting PCFG extensions.

Table 5 reports the accuracy results for all models on section 0 (*devset*) of the treebank. The accuracy results for the vanilla PCFG are approximately 10% lower than reported by (Charniak, 1996) for English demonstrating that parsing MH using the currently available treebank is a harder task. For all unlexicalized extensions learned from the transfromed treebanks, the resulting grammars show enhanced disambiguation capabilities and improved parsing accuracy. We observe that the vertical dimension contributes the most from both one-dimensional mod-

Name	Params	Description	Transforms used
DIST	$h = 0$	0-order Markov process	**Lexicalize(category), Linearize(distance)**
MRK	$h = 1$	1-order Markov process	**Lexicalize(category), Linearize(distance, neighbor)**
PA	$v = 1$	Parent Annotation	**Decorate(parent)**
DEF	$d = 1$	Definiteness feature percolation	**Decorate(definiteness)**

Table 4: **Implementing Different Parametrization Options using Transforms**

Implementation	(h, v, d)	F_{ALL} WP	$F_{\leq 40}$ WP	F_{ALL} WOP	$F_{\leq 40}$ WOP
PCFG	$(\infty, 0, 0)$	65.17	66.63	66.17	67.7
PA	$(\infty, 0, 1)$	70.6	71.96	70.96	72.18
DEF	$(\infty, 1, 0)$	67.53	68.78	68.82	70.06
PA+DEF	$(\infty, 1, 1)$	72.63	73.89	73.01	**74.11**

Table 5: **PCFG Two-Dimensional Extensions:** Accuracy results for parsing the *devest* (section 0)

els. A qualitative error analysis reveals that parent annotation strategy distinguishes effectively various kinds of distributions clustered together under a single category. For example, S categories that appear under TOP tend to be more flat than S categories appearing under SBAR (SBAR clauses typically generate a non-finite VP node under which additional PP modifiers can be attached).

Orthogonal morphological marking provide additional information that is indicative of the kind of dependencies that exist between a category and its various child constituents, and we see that the d dimension instantiated with *definiteness* not only contribute more than 2% to the overall parsing accuracy of a vanilla PCFG, but also contributes as much to the improvement obtained from a treebank already annotated with the vertical dimension. The contributions are thus additive providing preliminary empirical support to our claim that these two dimensions provide information that is complementary.

In our next set of experiments we evaluate the contribution of the depth dimension to extensions of the head-driven unlexicalized variety à la (Collins, 2003). We set our baseline at the $(0, 0, 0)$ coordinate and evaluate models that combine one, two and three dimensions of parametrization. Table 6 shows the accuracy results for parsing section 0 using the resulting models.

The first outcome of these experiments is that our new baseline improves on the accuracy results of a simple treebank PCFG. This result indicates that head-dependencies which play a role in determining grammatical structures in English are also instrumental for parsing MH. However, the marginal contribution of the head-driven variation is surprisingly low. Next we observe that for one-dimensional models the vertical dimension still contributes the most to parsing accuracy. However, morphological information represented by the depth dimension contributes more to parsing accuracy than information concerning immediately preceding sisters on the horizontal dimension. This outcome is consistent with our observation that the grammar of MH puts less significance on the position of constituents relative to one others and that morphological information is more indicative of the kind of syntactic relations that appear between them. For two-dimensional models, incorporating the depth dimension (orthogonal morphological marking) is better than not doing so, and relying solely on horizontal/vertical parameters performs slightly worse than the vertical/depth combination. The best performing model for two-dimensional head-driven extensions is the one combining vertical history and morphological depth. This is again consistent with the properties of MH highlighted in section 2 — parental information gives cues about the possible expansion on the current node, and morphological information indicates possible interrelation between child constituents that may be generated in a flexible order.

Our second set of experiments shows that a three-dimensional annotation strategy strikes the best balance between bias and variance and achieves the best accuracy results among all models. Different dimensions provide different sorts of information which are complementary, resulting in a model that is capable of generalizing better. The total error reduction from a plain PCFG is more than 20%, and our best result is on a par with those achieved for other languages (e.g., 75% for MSA).

Implementation	Params (h, v, d)	F_{ALL} WP	$F_{\leq 40}$ WP	F_{ALL} WOP	$F_{\leq 40}$ WOP
DIST	$(0, 0, 0)$	66.56	68.20	67.59	**69.24**
MRK	$(1, 0, 0)$	66.69	68.14	67.93	69.37
PA	$(0, 1, 0)$	68.87	70.48	69.64	70.91
DEF	$(0, 0, 1)$	68.85	69.92	70.42	**71.45**
PA+MRK	$(1, 1, 0)$	69.97	71.48	70.69	71.98
MRK+DEF	$(1, 0, 1)$	69.46	70.79	71.05	72.37
PA+DEF	$(0, 1, 1)$	71.15	72.34	71.98	**72.91**
PA+MRK+DEF	$(1, 1, 1)$	72.34	73.63	73.27	**74.41**

Table 6: **Head-Driven Three-Dimensional Extensions:** Accuracy results for parsing the *devest* (section 0)

Implementation	Params (h, v, d)	F_{ALL} WP	$F_{\leq 40}$ WP	F_{ALL} WOP	$F_{\leq 40}$ WOP
PCFG	$(\infty, 0, 0)$	65.08	67.31	65.82	68.22
PCFG+PA+DEF	$(\infty, 1, 1)$	72.26	74.46	72.42	**74.52**
DIST	$(0, 0, 0)$	66.33	68.79	67.06	69.47
PA+MRK+DEF	$(1, 1, 1)$	72.64	74.64	73.21	**75.25**

Table 7: **PCFG and Head-Driven Unlexicalized Models:** Accuracy Results for parsing the *testst* (section 12)

Figure 8 shows the $F_{All}(WOP)$ results for all models we implemented. In general, we see that for parsing MH higher dimensionality is better. Moreover, we see that for all points on the $(v, h, 0)$ plain the corresponding models on the $(v, h, 1)$ plain always perform better. We further see that the contribution of the depth dimension to a parent annotated PCFG can compensate, to a large extent on the lack of head-dependency information. These accumulative results, then, provide empirical evidence to the importance of morphological and morpho-syntactic processes such as definiteness for syntactic analysis and disambiguation as argued for in section 2.

We confirm our results on the *testset* and report in table 7 our results on section 12 of the treebank. The performance has slightly increased and we obtain better results for our best strategy. We retain the high error-reduction rate and propose our best result, 75.25% for sentences of length ≤ 40, as an empirically established string baseline on the performance of treebank grammars for MH.

6 Related Work

The MH treebank (Sima'an et al., 2001), a morphologically and syntactically annotated corpus, has been successfully used for various NLP tasks such as morphological disambiguation, POS tagging (Bar-Haim et al., 2007) and NP chunking (Goldberg et al., 2006). However its use for statistical parsing has been more scarce and less successful. The only previous studies attempting to parse MH we know of are (Sima'an et al., 2001), applying a variation of the DOP tree-gram model to 500 sentences, and (Tsarfaty, 2006), using a treebank PCFG in an integrated system for morphological and syntactic disambiguation.[9] The adaptation of state-of-the-art parsing models to MH is not immediate as the flat variable structures of phrases are hard to parse and a plentiful of morphological features that would facilitate disambiguation are not exploited by currently available parsers. Also, the MH treebank is much smaller than the ones for, e.g., English (Marcus et al., 1994) and Arabic (Maamouri and Bies, 2004), making it hard to apply data-intensive methods such as the all-subtrees approach (Bod, 1992) or full lexicalization (Collins, 2003). Our best performing model incorporates three dimensions of parametrization and our best result (75.25%) is similar to the one obtained by the parser of (Bikel, 2004) for Modern Standard Arabic (75%) using a fully lexicalized model and a training corpus about three times as large as our newest MH treebank.

This work has shown that devising an adequate baseline for parsing MH requires more than simple category-splits and sophisticated head-driven extensions, and our results provide preliminary evidence for the variation in performance of different parametrization strategies relative to the properties and structure of a given language. The comparison with parsing accuracy for MSA suggests that parametrizing an orthogonal depth dimension may be able to compensate, to some extent, on the lack of sister-dependencies, lexical information, and perhaps even the lack of annotated data, but establishing empirically its contribution to parsing MSA is a matter for further research. In the future we intend to further investigate the significance of the depth dimension by extending our models to include more morphological features, more variation in the pa-

[9]Both studies acheived between 60%–70% accuracy, however the results are not comparable to our study because of the use of different training sets, different annotation conventions, and different evaluation schemes.

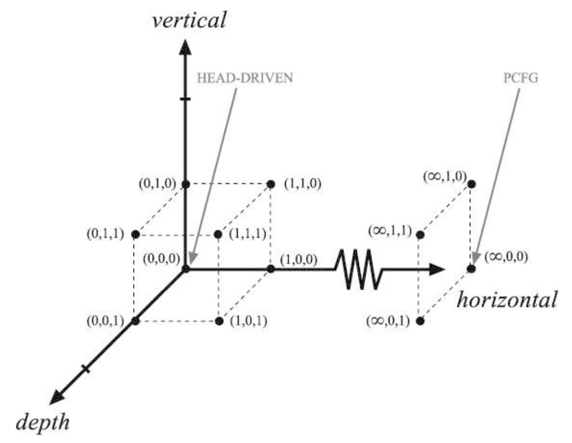


Figure 7: **All Models:** Locating Unlexicalized Parsing Models in a Three-Dimensional Parametrization Space

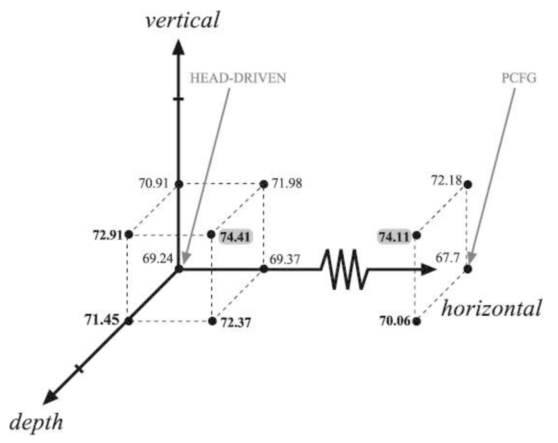


Figure 8: **All Results:** Parsing Results for Unlexicalized Models in a Three-Dimensional Parametrization Space

rameter space, and applications to more languages.

7 Conclusion

Morphologically rich languages introduce a new dimension into the expansion possibilities of a non-terminal node in a syntactic parse tree. This dimension is orthogonal to the vertical (Collins, 2003) and horizontal (Johnson, 1998) dimensions previously outlined by Klein and Manning (2003), and it cannot be collapsed into any one of the previous two. These additional dependencies exist alongside the syntactic head dependency and are attested using morphosyntactic phenomena such as long distance agreement. We demonstrate using syntactic definiteness in MH that incorporating morphologically marked features as a third, orthogonal dimension for annotating syntactic categories is invaluable for weakening the independence assumptions implicit in a treebank PCFG and increasing the model's disambiguation capabilities. Using a three-dimensional model we establish a new, stronger, lower bound on the performance of unlexicalized parsing models for Modern Hebrew, comparable to those achieved for other languages (Czech, Chinese, German and Arabic) with much larger corpora.

Tuning the dimensions and value of the parameters for learning treebank grammars is largely an empirical matter, and we do not wish to claim here that a three-dimensional annotation strategy is the best for any given language. Rather, we argue that for different languages different optimal parametrization strategies may apply. MH is not a free-word-order language in the canonical sense, and our qualitative analysis shows that all dimensions contribute to the models' disambiguation capabilities. Orthogonal dimensions provide complementary information that is invaluable for the parsing process to the extent that the relevant linguistic phenomena license grammatical structures in the language. Our results point out a principled way to quantitatively characterizing differences between languages, thus guiding the selection of parameters for the development of annotated resources, custom parsers and cross-linguistic robust parsing engines.

Acknowledgments We thank the Knowledge Center for Processing Hebrew and Dalia Bojan for providing us with the newest version of the MH treebank. We are particularly grateful to the development team of version 2.0, Adi Mile'a and Yuval Krymolowsky, supervised by Yoad Winter for continued collaboration and technical support. We further thank Felix Hageloh for allowing us to use the software resulting from his M.Sc. thesis work. We also like to thank Remko Scha, Jelle Zuidema, Yoav Seginer and three anonymous reviewers for helpful comments on the text, and Noa Tsarfaty for technical help in the graphical display. The work of the first author is funded by the Netherlands Organization for Scientific Research (NWO), grant number 017.001.271, for which we are grateful.

References

S. Abney. 1997. Stochastic Attribute-Value Grammars. *Computational Linguistics*, 23 (4):597–618.

R. Bar-Haim, K. Sima'an, and Y. Winter. 2007. Part-of-Speech Tagging of Modern Hebrew Text. *Journal of Natural Language Engineering*.

D. Bikel and D. Chiang. 2000. Two Statistical Parsing Models Applied to the Chinese Treebank. In *Second Chinese Language Processing Workshop*, Hong Kong.

D. Bikel. 2004. Intricacies of Collins' Parsing Model. *Computational Linguistics*, 4(30).

R. Bod. 1992. Data Oriented Parsing. In *Proceedings of COLING*.

E. Charniak. 1996. Tree-Bank Grammars. In *AAAI/IAAI, Vol. 2*, pages 1031–1036.

E. Charniak. 1997. Statistical Parsing with a Context-Free Grammar and Word Statistics. In *AAAI/IAAI*, pages 598–603.

M. Collins, J. Hajic, L. Ramshaw, and C. Tillmann. 1999. A Statistical Parser for Czech. In *Proceedings of ACL*, College Park, Maryland.

M. Collins. 2003. Head-Driven Statistical Models for Natural Language Parsing. *Computational Linguistics*, 29(4).

G. Danon. 2001. Syntactic Definiteness in the Grammar of Modern Hebrew. *Linguistics*, 6(39):1071–1116.

A. Dubey and F. Keller. 2003. Probabilistic Parsing for German using Sister-Head Dependencies. In *Proceedings of ACL*.

Y. Goldberg, M. Adler, and M. Elhadad. 2006. Noun Phrase Chunking in Hebrew: Influence of Lexical and Morphological Features. In *Proceedings of COLING-ACL*.

F. Hageloh. 2007. Parsing using Transforms over Treebanks. Master's thesis, University of Amsterdam.

M. Johnson. 1998. PCFG Models of Linguistic Tree Representations. *Computational Linguistics*, 24(4):613–632.

R. Kaplan and J. Bresnan. 1982. Lexical-Functional Grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, Cambridge, MA. The MIT Press.

D. Klein and C. Manning. 2003. Accurate Unlexicalized Parsing. In *Proceedings of ACL*, pages 423–430.

Y. Krymolowski, Y. Adiel, N. Guthmann, S. Kenan, A. Milea, N. Nativ, R. Tenzman, and P. Veisberg. 2007. Treebank Annotation Guide. MILA, Knowledge Center for Hebrew Processing.

M. Maamouri and A. Bies. 2004. Developing an Arabic Treebank: Methods, Guidelines, Procedures, and Tools. In *Proceedings of COLING*.

M. Marcus, G. Kim, M. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. 1994. The Penn Treebank: Annotating Predicate-Argument Structure.

T. Matsuzaki, Y. Miyao, and J. Tsujii. 2005. Probabilistic CFG with Latent Annotations. In *Proceedings of ACL'05*.

N. Melnik. 2002. *Verb-Initial Constructions in Modern Hebrew*. Ph.D. thesis, Berkeley University of California.

S. Petrov, L. Barrett, R. Thibaux, and D. Klein. 2006. Learning Accurate, Compact, and Interpretable Tree Annotation. In *Proceedings of ACL-COLING*, pages 433–440, Sydney, Australia, July.

D. Prescher. 2005. Head-Driven PCFGs with Latent-Head Statistics. In *In Proceedings of the International Workshop on Parsing Technologies*.

I. A. Sag, T. Wasow, and E. M. Bender. 2003. *Syntactic Theory: A Formal Introduction*. CSLI Publications, address, second edition.

H. Schmidt. 2004. Efficient Parsing of Highly Ambiguous Context-Free Grammars with Bit Vectors. In *Proceedings of COLING*, Geneva, Switzerland.

K. Sima'an, A. Itai, Y. Winter, A. Altman, and N. Nativ. 2001. Building a Tree-Bank of Modern Hebrew Text. In *Traitment Automatique des Langues*.

R. Tsarfaty. 2006. Integrated Morphological and Syntactic Disambiguation for Modern Hebrew. In *Proceeding of SRW COLING-ACL*.

S. Wintner. 2000. Definiteness in the Hebrew Noun Phrase. *Journal of Linguistics*, 36:319–363.

Data-Driven Dependency Parsing across Languages and Domains: Perspectives from the CoNLL 2007 Shared Task

Joakim Nivre
Växjö University, School of Mathematics and Systems Engineering
Uppsala University, Department of Linguistics and Philology
E-mail: nivre@msi.vxu.se

Abstract

The Conference on Computational Natural Language Learning features a shared task, in which participants train and test their learning systems on the same data sets. In 2007, as in 2006, the shared task has been devoted to dependency parsing, this year with both a multilingual track and a domain adaptation track. In this paper, I summarize the main findings from the 2007 shared task and try to identify major challenges for the parsing community based on these findings.

1 Introduction

The annual Conference on Computational Natural Language Learning (CoNLL) has for the past nine years organized a *shared task*, where participants train and test their learning systems on the same data sets. In 2006, the shared task was multilingual dependency parsing, where participants had to train and test a parser on data from thirteen different languages (Buchholz and Marsi, 2006). In 2007, the task was extended by adding a second track for (monolingual) domain adaptation.

The CoNLL 2007 shared task on dependency parsing featured two tracks:

- In the *multilingual track*, the task was to train a parser using labeled data from Arabic, Basque, Catalan, Chinese, Czech, English, Greek, Hungarian, Italian, and Turkish.
- In the *domain adaptation track*, the task was to adapt a parser for English news text to other domains using unlabeled data from the target domains: biomedical and chemical abstracts, parent-child dialogues.[1] In the *closed class*, the base parser had to be trained using the English training set for the multilingual track and no external resources were allowed. In the *open class*, any base parser could be used and any external resources were allowed.

Both tracks used the same column-based format for labeled data with six input columns and two output columns for each word of a sentence:

- Input: word-id, word form, lemma, coarse part of speech, fine part-of-speech, morphosyntactic features.
- Output: head (word-id), dependency label.

The main evaluation metric for both tracks was the *labeled attachment score* (LAS), i.e., the percentage of words that have been assigned the correct head and dependency label. For more information about the setup, see Nivre et al. (2007)

In this paper, I will summarize the main findings from the CoNLL 2007 shared task, starting with a characterization of the different approaches used (section 2), and moving on to the most interesting results in the multilingual track (section 3) and the domain adaptation track (section 4). Finally, based on these findings, I will try to identify some important challenges for the wider parsing community (section 5).

[1]The biomedical domain was the development domain, which means that a small labeled development set was available for this domain. The final testing was only done on chemical abstracts and (optionally) parent-child dialogues.

2 Approaches

In total, test runs were submitted for twenty-three systems in the multilingual track, and ten systems in the domain adaptation track (six of which also participated in the multilingual track). The majority of these systems used models belonging to one of the two dominant approaches in data-driven dependency parsing in recent years (McDonald and Nivre, 2007):

- In *graph-based models*, every possible dependency graph for a given input sentence is given a score that decomposes into scores for the arcs of the graph. The optimal parse can be found using a spanning tree algorithm (Eisner, 1996; McDonald et al., 2005).
- In *transition-based models*, dependency graphs are modeled by sequences of parsing actions (or transitions) for building them. The search for an optimal parse is often deterministic and guided by classifiers (Yamada and Matsumoto, 2003; Nivre, 2003).

The majority of graph-based parsers in the shared task were based on what McDonald and Pereira (2006) call the first-order model, where the score of each arc is independent of every other arc, but there were also attempts at exploring higher-order models, either with exact inference limited to projective dependency graphs (Carreras, 2007), or with approximate inference (Nakagawa, 2007). Another innovation was the use of k-best spanning tree algorithms for inference with a non-projective first-order model (Hall et al., 2007b).

For transition-based parsers, the trend was clearly to move away from deterministic parsing by adding a probability model for scoring a set of candidate parses typically derived using a heuristic search strategy. The probability model may be either conditional (Duan et al., 2007) or generative (Titov and Henderson, 2007).

An interesting way of combining the two main approaches is to use a graph-based model to build an ensemble of transition-based parsers. This technique, first proposed by Sagae and Lavie (2006), was used in the highest scoring system in both the multilingual track (Hall et al., 2007a) and the domain adaptation track (Sagae and Tsujii, 2007).

3 Multilingual Parsing

The ten languages involved in the multilingual track can be grouped into three classes with respect to the best parsing accuracy achieved:

- Low (LAS = 76.3–76.9):
  Arabic, Basque, Greek
- Medium (LAS = 79.2–80.2):
  Czech, Hungarian, Turkish
- High (LAS = 84.4–89.6):
  Catalan, Chinese, English, Italian

To a large extent, these classes appear to be definable from typological properties. The class with the highest top scores contains languages with a rather impoverished morphology. Medium scores are reached by the two agglutinative languages, Hungarian and Turkish, as well as by Czech. The most difficult languages are those that combine a relatively free word order with a high degree of inflection. Based on these characteristics, one would expect to find Czech in the last class. However, the Czech training set is four times the size of the training set for Arabic, which is the language with the largest training set of the difficult languages. On the whole, however, training set size alone is a poor predictor of parsing accuracy, which can be seen from the fact that the Italian training set is only about half the size of the Arabic one and only one sixth of Czech one. Thus, there seems to be a need for parsing methods that can cope better with richly inflected languages.

4 Domain Adaptation

One result from the domain adaptation track that may seem surprising at first was the fact that the best closed class systems outperformed the best open class systems on the official test set containing chemical abstracts. To some extent, this may be explained by the greater number of participants in the closed class (eight vs. four). However, it also seems that the major problem in adapting existing, often grammar-based, parsers to the new domain was not the domain as such but the mapping from the native output of the parser to the kind of annotation provided in the shared task data sets. In this respect, the closed class systems had an advantage by having been trained on exactly this kind of annotation. This result serves to highlight the fact that domain adaptation, as well as the integration of grammar-based and data-driven methods, often involves transformations between different kinds of linguistic representations.

The best performing (closed class) system in the domain adaptation track used a combination of co-learning and active learning by training two different parsers on the labeled training data, parsing the unlabeled domain data with both parsers, and adding parsed sentences to the training data only if the two parsers agreed on their analysis (Sagae and Tsujii, 2007). This resulted in a LAS of 81.1 on the test set of chemical abstracts, to be compared with 89.0 for the English test set in the multilingual track.

5 Conclusion

Based on the results from the CoNLL 2007 shared task, it is clear that we need to improve our methods for parsing richly inflected languages. We also need to find better ways of integrating parsers developed within different frameworks, so that they can be reused effectively for, among other things, domain adaptation. More generally, we need to increase our knowledge of the multi-causal relationship between language characteristics, syntactic representations, and parsing and learning methods. In order to do this, perhaps we also need a shared task at the International Conference on Parsing Technologies.

Acknowledgments

I want to thank my fellow organizers of the shared task, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret, who are also co-authors of the longer paper on which this paper is partly based (Nivre et al., 2007). I am also indebted to all the people who have contributed to the shared task by providing data or participating.

References

S. Buchholz and E. Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proc. of CoNLL*, 149–164.

X. Carreras. 2007. Experiments with a high-order projective dependency parser. In *Proc. of EMNLP-CoNLL (Shared Task)*.

X. Duan, J. Zhao, and B. Xu. 2007. Probabilistic parsing action models for multi-lingual dependency parsing. In *Proc. of EMNLP-CoNLL (Shared Task)*.

J. M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proc. of COLING*, 340–345.

J. Hall, J. Nilsson, J. Nivre, G. Eryigit, B. Megyesi, M. Nilsson, and M. Saers. 2007a. Single malt or blended? A study in multilingual parser optimization. In *Proc. of EMNLP-CoNLL (Shared Task)*.

K. Hall, J. Havelka, and D. Smith. 2007b. Log-linear models of non-projective trees, k-best MST parsing and tree-ranking. In *Proc. of EMNLP-CoNLL (Shared Task)*.

R. McDonald and J. Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proc. of EMNLP-CoNLL*.

R. McDonald and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proc. of EACL*, 81–88.

R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proc. of HLT/EMNLP*, 523–530.

T. Nakagawa. 2007. Multilingual dependency parsing using gibbs sampling. In *Proc. of EMNLP-CoNLL (Shared Task)*.

J. Nivre and J. Nilsson. 2005. Pseudo-projective dependency parsing. In *Proc. of ACL*, 99–106.

J. Nivre, J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proc. of EMNLP-CoNLL (Shared Task)*.

J. Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proc. of IWPT*, 149–160.

K. Sagae and A. Lavie. 2006. Parser combination by reparsing. In *Proc. of HLT-NAACL (Short Papers)*, 129–132.

K. Sagae and J. Tsujii. 2007. Dependency parsing and domain adaptation with LR models and parser ensembles. In *Proc. of EMNLP-CoNLL (Shared Task)*.

I. Titov and J. Henderson. 2007. Fast and robust multilingual dependency parsing with a generative latent variable model. In *Proc. of EMNLP-CoNLL (Shared Task)*.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proc. of IWPT*, 195–206.

Author Index

ACL 2007

PRAGUE

ISBN 978-1-932432-90-9

ISBN 978-1-932432-90-9

52500

9 781932 432909